FALL 2023

CSCI 6470

ALGORITHMS

CSCI 6470

Algorithms

Prof: Dr. Liming Cai

Part 1: Introduction

★ Topics to be covered:

- Measuring computational complexities
- Crafting recursive algorithms
- Time complexity of recursive algorithms.

Algorithms? Finite steps to solve a problem

What is complexity algorithms?

- CPU time, memory used → time and space

Why measuring complexity?

- Practical usefulness

How to measure?

- Math notions required

★ How do we know an alg's complexity is acceptable?

- Use absolute standards (for ex. $O(n)$ we will accept, but not $O(n^2)$)
- Compare w/ other algorithms

★ Why are math notions needed?

- Diff hardware, system platforms, languages, etc
- Amt of data to test issues if based on testing.

} Math notions would make those issues disappear

(ex)

```
A = B + C
A = 0
for (i = 1 to n) do
A = A + i
```

	Assembly code	⇔	Machine code
⇒	LOAD R_1, B		001 001
	LOAD R_2, C		001 010
	ADD R_1, R_2		100 001
	STORE R_1, A		111 ...

- Count the number of basic operations
- And in terms of input size

0. Measuring complexities

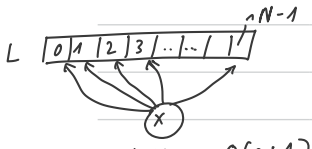
length of array to search

```
function search (L, x, N);
{
  i = 0;                                  — 1
  while (L[i] != x) AND (i < N)           (3+1) × (N+1)
  {
    i = i + 1;                            2 × N
  }

  if (i < N)
    return (i)                            — 2
  else
    return (o)                            — 1
}
```

⇒ Total time: $4 + 4(N+1) + 2N$

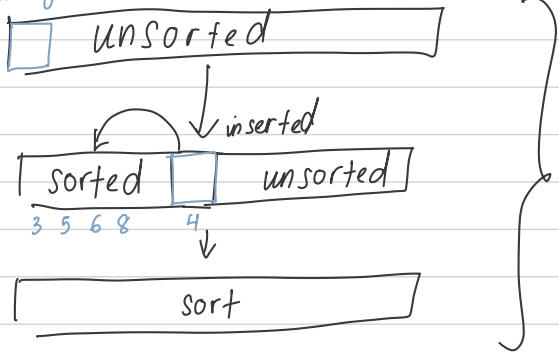
$= 6N + 8$



Worst time: $O(n+1)$

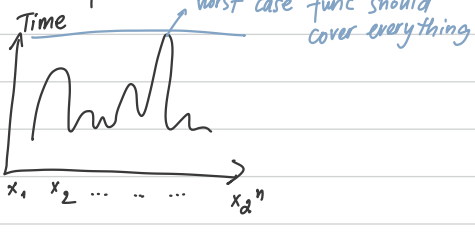
08/21

always sorted - 1st elem



- write algs

$f(n) = 3n^2 - 20n + 100$ $\quad |x_1| = n$



Definition of big O:

Let $f(n)$ and $g(n)$ be 2 functions in n.

$f(n) = O(g(n))$ if there exists a constant c and n_0 such that

$$f(n) \leq cg(n)$$

when $n \geq n_0$

ex: find big O: $f(n) = 3n^2 - 20n + 100$?

$3n^2 - 20n + 100 \leq 3n^2 + 100$

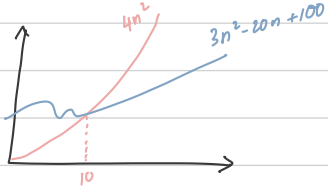
$\leq 3n^2 + n^2$ when $n \geq 10$

$\leq 4n^2$

$\Rightarrow$ We have found $c = 4$ and $n_0 = 10$ such that $f(n) \leq cn^2$.

Therefore, $f(n) = O(n^2)$

$\Rightarrow$ "$f(n)$ is big-O of n^2"



ex: $3n^3 + 2n - 6 = O(?)$

$3n^3 + 2n - 6 \leq \ldots \leq Cn^3$

$\leq 3n^3 + 2n^3 \quad (n \leq n^3)$

$\leq \boxed{5n^3}$

$\Rightarrow c = 5$

ex: $3n\log_2 n + 5n + 7\log_2 n = O(?)$

$3n\log_2 n + 5n + 7\log_2 n \leq 3n\log_2 n + 5n\log_2 n + 7n\log_2 n \leq 15n\log_2 n$

$c = 15$

$n_0 = 2$ (b/c n_0 has to be > 1 b/c $\log_2 1 = 0$)

When $T(2)$

$\Rightarrow 3 \cdot 2 \cdot 1 + 5 \cdot 2 + 7 \cdot 1 \leq 30 \Rightarrow 23 \leq 30$

$\rightarrow$ Yes

ex: $2^{2n} + 3 \cdot 2^n = O(?)$

Suppose $2^{2n} + 3 \cdot 2^n \leq c2^n$ Trial

$2^n \cdot 2^n + 3 \cdot 2^n \leq c2^n$

$2^n + 3 \leq c$

n has to be bounded by c

$\Rightarrow$ WRONG ✗

$2^{\frac{1}{2}n} = \left(2^{\frac{1}{2}}\right)^n = \sqrt{2}^n = O(2^n)$

ex: $5\ln n + 7\log_{10} n + 2\log_2 n = O(?)$

$\log_{10} n = \frac{\log_2 n}{\log_2 10}$

$\log_{10} n \leq \left(\frac{1}{\log_2 10}\right) \log_2 n$ (c)

$\log_2 n = (\log_2 10) \log_{10} n$

$\log_{10} n = \frac{\log n}{\log 10}$

$\log_{10}{}^n < \log_2{}^n$

smaller base → larger log

smaller base → larger big O

log comparision rule

$\Rightarrow \log_a b = \frac{\log_c b}{\log_c a}$ ★ (c)

$\ln n = \log_e n = \frac{\log_2 n}{\log_2 e}$ ★

Work

$5\ln(n) + 7\log_{10}(n) + 2\log_2(n)$ $= k\log_2 n$ ..... $k\log_2 n$

$= 5\ln(n) + \frac{7\log_2 n}{\log_2 10} + 2\log_2(n)$

$= 5\ln(n) + \left(\frac{7}{\log_2 10} + 2\right)\log_2 n$

$= \frac{5\log_2(n)}{\log_2 e} + \ldots\ldots = \log_2 n \left(\frac{5}{\log_2 e} + \frac{7}{\log_2 10} + 2\right)$

$f(n) \leq c\,(gn)$ for $n \geq n_0$

Notes: • More than 1 way to derive big O

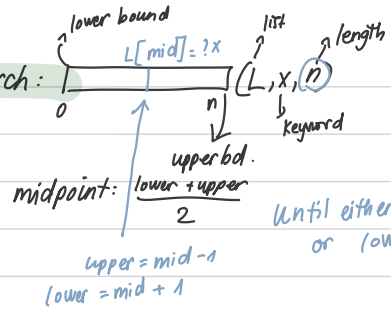
① $3n^2 - 20n + 100 \leq 4n^2$, $n \geq n_0 = 10$

② $3n^2 - 20n + 100 \leq 10^2 n^2$, $n \geq n_0 = 1$

• c and n_0 can always be replaced with larger number

• asymptotic, almost like $\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

☆ Exercise: Binary search:




```
low = 1        high = n
while low < high do {
    mid = (low + height)/2
    if L[mid] = x
        return mid
    else {
        if L[mid] < x    low = mid + 1
        else high = mid - 1;
    }
```

Binary

length = height - low

length / 2

$\rightarrow \frac{length}{2} \rightarrow \frac{length}{4} \rightarrow \frac{length}{8}$

n_{halved} in each recursive call

until $\frac{n}{2^k}$ → no. of iterations

But worst case we go thru and nothing to divide (no keyword in L)

worst case = number of iterations = $\frac{n}{2^k} = 1$

$\frac{n}{2^k} = 1$ $\quad n = 2^k$

$\log_2 n = k$

Running time: $(\log_2 n) + 1 \leq \log_2 n$ $\quad n_0 = 2$

Just more work to understand, not required

$f(n)$ = no. of operations of binary search

$\Rightarrow f(n) = 3k$ (as binary has more no. of operations)

$k = 3 \log(n)$ $\Rightarrow$ $f(n) \leq 3 \log(n)$ ↳ $c = 3$, $n_0 = 0$

08/24/23

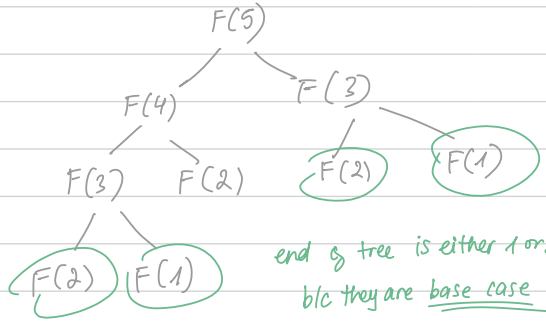
Recursion

- Recursion: using the same process

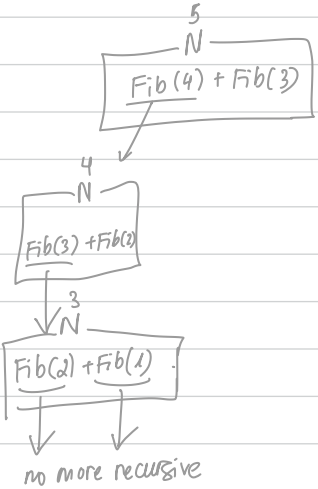
⇒ a process to solve a problem in terms of the same process.

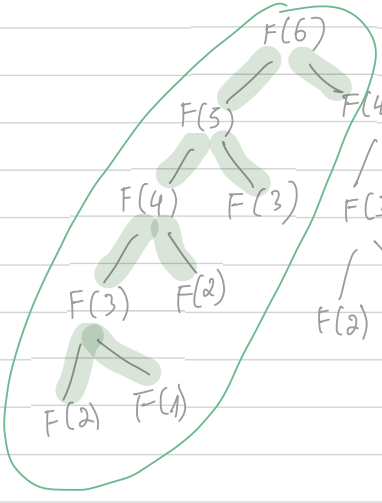
executing a recursive alg / program:

```
function Fib(N)
    if (N=1)
    if (N=1) return (0)
    if (N=2) return (1)
    else
        return (Fib(n-1) + Fib(n-2))
```



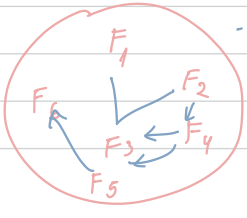
end of tree is either 1 or 2 b/c they are base case





we remember already computed values, so actually the green branch is the only one executed.

⇒ $F_N = F_{N-1} + F_{N-2}$



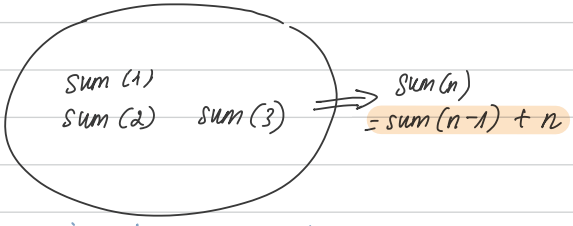
- relationships between elements in the set
- structured

graphs:
nodes + edge

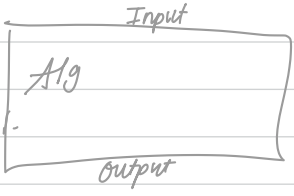
★ derive recursive pattern

up to n

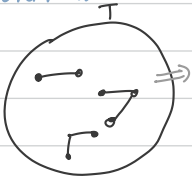
$1 + 2 + 3 + \dots + n$



★ craft a recursive alg:



★ Graph overview



① a single node $\in T$

② if $t \in T \rightarrow$ is tree

vertex v & t, $n \in t$

$\Rightarrow t \cup \{(u,v)\} \in T$

t + edge (u,v) is tree

Overview stuff, useful for recursion

★ The set of lists

finite list

a list $(a_1, a_2, \dots, a_n)$ $\quad n \geq 0$ (starting from empty list)

① ( ) is a list ($\in L$)

② if l is a list
then $l.(a)$ is a list, a is the element

★ Sorted list

$(a_1, a_2, \dots, a_n)$ $\quad a_i < a_{i+1}, i = 1$ $\quad n \geq 0$

① ( ) is sorted list

② if $l = (a_i, \dots, a_k)$ is in SL
then $l.(a)$ is also in SL
where $a \geq a_i$, $i = 1 \dots k$

n items; # of arrangements of them in order

A B C ⇒ ABC
ACB
CBA

ex) A B C D

Let $P(N)$ to be # of arrangements of elements

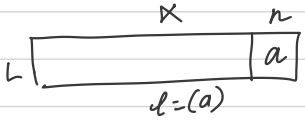
$$P(N) = \begin{cases} nP(n-1) & n \geq 2 \\ 1 & n = 1 \end{cases}$$

ex) n : items to chose k: items # of choices : $\binom{n}{k}$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Actualy, why ?

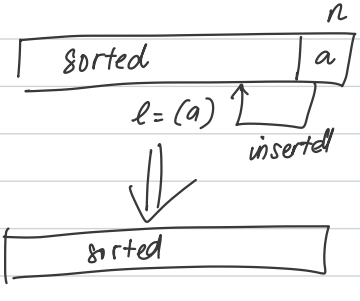
★ Deriving recursive pattern for linear search




```
Linear Search (L, x, n)
if n = 0
    return (-1)        → not found
else
    if L[n] = x
    return (n)
    else
```

Linear search (L, x, n-1)

★ Deriving recursive pattern for insertion sort:




```
Insertion Sort (L, n)
    if (n>1)
    Insertion (L, n-1)
    Insertion (L, n)
```

Quiz on Monday

- Analysis of time complexity iterative algorithm
- Big O
  func = O(?)
- Understand recursive alg

08/27/23 Quiz Review

* Measuring computational complexity

- complexity < time – CUP time
  space – memory

linear search O(n)

Basic operation

```
function search (L, x, N);
{
  i = 0;                          — 1
  while (L[i] != x) AND (i < N)   = 4x(n+1)
  {
    i = i + 1;                    ⇒ 2xN
  }

  if (i < N)                      — 1
    return (i)                    — 1
  else
    return (o)                    — 1
}
```

still considering i = n cause it's when the loop breaks

$= 4 + 4(n+1) + 2n$

$\Rightarrow \boxed{6N + 8}$

Fibonacci sequences → calculating nth number

```
function dosomething (N);
{
  x = 0;                          — 1
  y = 1;                          — 1
  i = 1;                          — 1
  while (i < N)                   1 x N ⇒ N
  {
    i = i + 1;                    — 2N
    t = x;                        — 1N
    x = y;                        — 1N
    y = t + x;                    — 2N
  }
  if (N=1)                        — 1
    return (x);                   — 1
  else
    return (y);                   — 1
}
```

3

N(2+1+1+2) ⇒ 6N

3

$6N + N = 7N$

$\Rightarrow \boxed{7N + 6}$

* Insertion sort $O(N^2)$

$f(n) = O(g(n))$ if there exist constant c and n_0 such that
$f(n) \leq cg(n)$ when $n \geq n_0$

Hierarchy of Big O:

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n \cdot \log n)$
- $O(n^2)$
- $O(n^c)$
- $O(n!)$

(ex) $3x^2 + 25$

$3x^2 + 25 \leq 3x^2 + 25x^2$

$\leq \underbrace{28x^2}_{c}$ when $x \geq \underbrace{1}_{n_0}$

(ex) $4x^3 + 7x^2 + 12$

$4x^3 + 7x^2 + 12 \leq 4x^3 + 7x^3 + 12x^3 \quad (x^2, x < x^3)$

$\leq \underbrace{23x^3}_{c}$, $n_0 = 1$

(ex) prove $3n^2 + 8n + 4 \quad O(n^2)$

$3n^2 + 8n + 4 \leq 3n^2 + 8n^2 + 4n^2 \leq \underbrace{15n^2}_{c}$ when $n \geq 1 \Rightarrow \boxed{n_0 = 1}$

(ex) Prove $n^3 + 20n^2 + 2 \neq O(n)$ when $n = 10$

$n^3 + 20n^2 + 2 \leq n + 20n + 2n$

plug in $n = 10$

(2) $3002 \leq 230 \Rightarrow$ WRONG!

(ex)

$\log_4(4n^2 + 5n + 8) \in O(\log_4(n))$

$\log_4(4n^2 + 5n + 8) \leq \log_4(4n^2 + 5n^2 + 8n^2)$

$\leq \log_4(17n^2)$

$\leq \log_4(17) + \log_4(n^2)$ } log rule

$\leq \log_4(17) + 2\log_4(n)$ ← make first term a number $\times \log_n$

$\leq \log_4(n) + 2\log_n(n)$ when $n \geq 17$

$\leq \boxed{3}\log_4(n)$

c

$\boxed{n_0 = 17}$

$\log_4(17) \leq \log_4(n)$ → can use this

$4^{\log_4(17)} \leq 4^{\log_4(n)}$

$17 \leq n$

(ex) $2n^2 + 3n + 4 \in O(n^2)$

$2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2$

$\leq 9n^2$

c when $\boxed{n_0 = 1}$

$3n \leq 3n^2$

$1 \leq n$

$4 \leq 4n^2$

$1 \leq n^2$

$1 \leq n$

☆ Recursion

- Base case (always before recursive step)
- Recursive step

☆ Old midterm exam practice

① $20n\log_2 n - n^2 + 3n = O(n\log_2 n)$ F (must be n^2)

② $345 = O(n)$ T

③ $3n^2 - 2000 n\log_2 n = O(n\log_2 n)$ F

ex $T(n) = 3n\log_2 n + 4n - 2$. Prove $T(n) = O(n\log_2 n)$. Find c, k (n_0)

$T(n) = 3n\log_2 n + 4n - 2 \leq 3n\log_2 n + 4n$

$\leq 3n\log_2 n + 4n\log_2 n$

$\leq 7n\log_2 n$

$c = 7$; $n_0 = k = 2$

$T(1) = 3.1.\log_2(1) + 4.(1) - 2 = 2 \not\leq 7.1\log(1)$

$T(2) = 3(2).\log_2(2) + 4.(2) - 2 = 6\log_2(2) + 6$

$= 12$

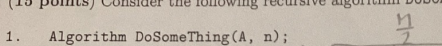
$\leq 7.2.\log_2(2) = 14$

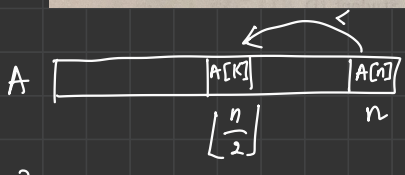
ex

3. (15 points) Consider the following recursive algorithm DoSomeThing. You may assume $n \geq 1$.

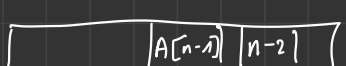
15

```
1.  Algorithm DoSomeThing(A, n);
2.  C1  if n==1 return A[1];
3.      else
4.          k = L n/2 J                  // pair of symbols "L, J" represents the floor function
5.  C2      if A[n] > A[k]
6.          A[k] = A[n];
7.      DoSomeThing(A, n-1);  // this statement is in parallel with the 'if',
                              // not within 'if'
```





⇒



if A has 1 element, return that element

else $k = \lfloor \frac{n}{2} \rfloor$

last element > mid element

assign last (larger) elem to mid

and keep doing until the list has only 1 elem

⇒ return that elem

⇒ Return max number between n and 1

⇒ Base case: $n = 1$, $T(n) = C_1$

Recursive case: $T(n) \leq T(n-1) + C_2$ for $n \geq 2$

prove

(ex) $2n^2 + 3n + 4 \in O(n^4)$

$2n^2 + 3n + 4 \leq 2n^4 + 3n^4 + 4n^4$

$\leq 9n^4$

$c = 9$, $n_0 = 1$

$2n^2 \leq 2n^4$	$3n \leq 3n^4$	$4 \leq 4n^4$
$1 \leq n^2$	$1 \leq n^4$	$1 \leq n^4$
$n \geq 1$	$1 \leq n^2$	$n \geq 1$
	$n \geq 1$	

(ex) Recursion in linear search (L, x, n)

```
if (n=0) return "Not found"
else
    if L(n) = x return (n)
    else return ( LinearSearch (L, x, n-1)
```

Fibonacci sequences → calculating nth number

```
function dosomething (N);
{
  x = 0;          — 1
  y = 1;          — 1
  i = 1;          — 1
  while (i < N)   1 x N => N
  {
    i = i + 1;    — 2(N-1)
    t = x;        — 1(N-1)
    x = y;        — 1(N-1)
    y = t + x;    — 2(N-1)
  }
  if (N=1)        — 1
    return (x);   — 1
  else
    return (y);   — 1
}
```

3 + (2N − 2 + N − 1 + N − 1 + 2N − 2) + 3 = 6N − 6

7N − 6 ⇒ 7N + 6 − 6 = 7N

08/27/23

More on Recursion Lectures

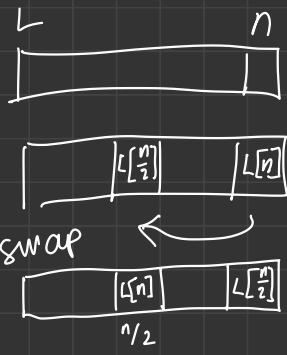
Quiz 1 Algorithms

(4)

4. (10 points) Given the following recursive algorithm,

```
function DoSomething(L, n); // L is a list indexed from 1 to n, n>=1
  if (n = 1)
     return (L[1]);
  else
    if (L[n] > L[n/2])
       swap(L[n/2], L[n]);  // assumed n can be evenly divided by 2
    return (DoSomething(L, n-1));
```

2 cases: max at left / at right } ⇒ output: max always left



① What's output of [2, 4, 8, 6, 9, 5]

index $\frac{n}{2}=3$: 2 4 (8) 6 9 (5) — n=6

$\frac{5}{2}=2$: 2 (4) 8 6 (9) 5 — n=5

$\frac{4}{2}=2$: 2 (9) 8 (6) 4 5 — n=4

$\frac{3}{2}=1$: (2) 9 (8) 6 4 5 — n=3

$\frac{2}{2}=1$: (8) (9) 2 6 4 5 — n=2

(9) 8 2 6 4 5

n = 1 ⇒ When n=1, return L[n] ⇒ return 1st elem

⇒ Output 9

⇒ Return the max element

9 8 (2) 3 10 (1) n=6

9 8 2 3 (10) 1 n=5

9 10 2 (3) 8 1 n=4

9 10 (2) 3 8 1 n=3

9 (10) 2 3 8 1 n=2

10 9 2 3 8 1

max elem at n=1

⇒ Return the max element in the list from n=1

② DoSomething is called n times on n input

③ What does this algo do? Return max element from input list

(3) ① $n^3+2n-6 = O(n^4)$ ⇒ True

② $10\cdot 2^n = O(n^{100})$ ⇒ False

↳ 2^n grows faster than n^{100}.

③ $2^n + 2\cdot 2^{\frac{n}{2}} = O(2^{\frac{n}{2}})$ ⇒ False

should be 2^n

④ $5\log_2 n + 5\ln n = O(\log_{10} n)$ ⇒ True

$5\log_2 n + 5\log_e n \in O(\log_{10} n)$ $(e \approx 2.71827)$

We don't care about base number in Big O for log.

⑤ $7\sqrt{n} = O((\log_2 n)^2)$ ⇒ False

$\sqrt{n}$ still grows faster than $(\log_2 n)^2$.

⇒ cannot be bounded by $((\log_2 n))^2$

Big O growth hierachy: small → high

1

$\log_{10}(n)$

$\log_2(n)$ ⇒ $\log n < \sqrt{n}$

$\sqrt{n}$ ⇒ $\log_n$ is less complex / grow more slowly than $\sqrt{n}$

n

$n\log_{10}(n)$

n^2

(1)

```
Function M(L, n); // L is a list indexed f      o n, n>=1
  m = L[1];                                          2
  k = 2;                                             1
  while (L[k] >= m) & (k <= n)  4 × n                4n
    k = k + 1;  2(n-1)                               2(n-1)
    m = L[k];   2(n-1)                               2(n-1)
  return (m);   1                                    1
```

$k = 2$ $k \le n$ $\Rightarrow n-1$ times but also n times (last condition before break)

$n = 4$	
$k = 2$	1
$k = 3$	1
$k = 4$	1
$k = 5$	1
	X

1) How many basic operations?

Total = $2 + 1 + 4n + 2(n-1) + 2(n-1) + 1$

$= 3 + 4n + 2n - 2 + 2n - 2 + 1$

$= \boxed{8n}$

2) What does this algo do?

Output the last number in the monotically increasing sequence from L[1]

★ Things to ask:
- Question 1: $4 \times n$?
- $\log_2 n > \log_{10} n$?

(2)

2. **(10 points)** This question is about to upper-bound a function.

(1) Let $f(n) = 4n + 2\log_2 n + 2$. Fill in the blank space: $c = 8$, $n_0 = 1$, such that

$f(n) \le cn$ when $n \ge n_0$

(2) Let $f(n) = 4n + 2\log_2 n + 2$. Fill in the blank space $c =$ 8, $n_0 = 2$ such that

$f(n) \le cn\log_2 n$ when $n \ge n_0$

(1) $4n + 2\log_2 n + 2 \le 4n + 2n + 2n$

$\le \underbrace{8n}_{c}$ when $\boxed{n_0 = 1}$

$T(0)$: $2 \le 0$ (False)

$T(1)$: $4 + 2 \le 8$ (True)

$\log_2 n \le n$

$1 \le n$

$\Rightarrow n \ge 1$

(2) $4n + 2\log_2 n + 2 \le 4n\log_2 n + 2n\log_2 n + 2n\log_2 n$

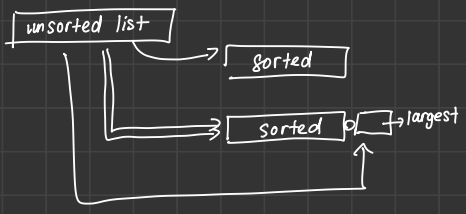
$\le \underbrace{8n\log_2 n}_{c}$ when $\boxed{n_0 = 2}$

★ Solve each inequality

$\begin{cases} n \le n\log_2 n \\ \log_2 n \le n\log_2 n \\ 1 \le n\log_2 n \end{cases} \Rightarrow \begin{cases} 1 \le \log_2 n \Rightarrow n \ge 2 \\ n \ge 1 \\ n \ge 2 \end{cases} \Rightarrow n \ge 2$

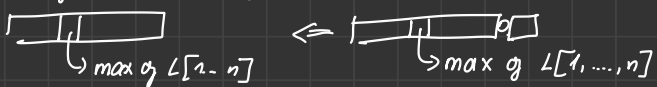
08/29/23

Selection Sort (Recursion)




```
func SelectionSort(L, n)
    if n > 1
    Findmax (L, n) // Find the max elem & swap it w/ L[n]
    SelectionSort (L, n-1)
```

★ idea of list (recursively)




```
if n > 1
    func FindMax(L, n)
    if (L[n-1] > L[n])
        swap(L[n-1], L[n])
```

might look like it's sorted but it's not

★ Complexity Analysis For Recursive Alg

① Formulate a recursive func T(n) for the alg.

ex: $T(n) = T(n-1) + 5$ (including base case)

② Solve T(n), typically using induction methods) to get non-recursive expression.

ex: $T(n) = 2n + 4n + 2$

③ show the big O for T(n)

Recursive algorithms:

★ Linear Search

```
function LinearSearch(L, x, n);                <---- T(n)
if (n = 0) return ("not found");               <----- c1
else
  if (L[n]=x) return (n);                      <----- c2
  else return (LinearSearch(L, x, n-1));       <----- c3 + T(n-1)
```

Let T(n) be time func for LinearSearch(L, x, n)

$T(n) = c_1$ or c_2 or $c_3 + T(n-1)$

$T(n) = c_3 + T(n-1)$

$T(0) = c_1$

Step 1: $T(n) = \begin{cases} c_3 + T(n-1) & n = 1 \\ c_1 & n = 0 \end{cases}$

Step 2:

$T(n) = T(n-1) + c_3$

$+ T(n-1) = T(n-2) + c_3$

$\vdots$

$+ T(1) = T(0) + c_3$

add all equations

$T(n) = T(0) + c_3 \times n$

$= c_1 + c_3 \times n$

get this before step 3

Step 3: $T(n) = O(n)$

b/c $c_1 + c_2 n \leq c_1 n + c_2 n$

$\leq \underbrace{(c_1 + c_2)}_{c} n$

$n_0 = 1$

★ Analyze worst case time complexity for recursive selection sort algorithm:

```
S(n) _ func FindMax (L, n)
c    _ if n > 1
  S(n-1)  Findmax (L, n-1)
  c'      if (L[n-1] > L[n])
              swap (L[n-1], L[n])
```

```
T(n) _ func SelectionSort (L, n)
       if n > 1 _ c
  S(n)   _____ Findmax (L, n)
  T(n-1) _____ SelectionSort (L, n-1)
```

★ Step 1

$S(n) = S(n-1) + c + c'$

$= S(n-1) + c$

$\Rightarrow S(n) = \begin{cases} S(n-1) + c & \text{when } n > 1 \\ c & n = 1 \end{cases}$

$T(n) = S(n) + T(n-1) + c$

plug in, add until $S(n) = S(n-1) + c$

$S(n-1) = S(n-2) + c$

$\vdots$

$S(2) = S(1) + c$

$\Rightarrow S(n) = S(1) + c(n-1)$

$\Rightarrow S(n) = \underbrace{S(1) - c}_{a} + cn$

$\Rightarrow S(n) = a + bn$

$S(n) = a + bn$

$\Rightarrow T(n) = T(n-1) + (a + bn) + c$ (combine $a + c$)

$= T(n-1) + bn + d$ when $n > 1$

$\Rightarrow T(n) = \begin{cases} T(n-1) + bn + d & , n > 1 \\ c & n = 1 \end{cases}$

★ Step 2:

$T(n) = T(n-1) + bn + d$

$T(n-1) = T(n-2) + b(n-1) + d$

$T(n-2) = T(n-3) + b(n-2) + d$

$\vdots$

$T(2) = T(1) + b \cdot 2 + d$

$T(n) = T(1) + b\left(n + (n-1) + (n-2) + \dots + 2\right) + (n-1)d$

From $T_n \to T_2$

$= c + b\left(\frac{n}{2}(n+1) - 1\right) + (n-1)d$

$= c + \frac{b}{2}n^2 + \frac{b}{2}n - b + nd - d$

$= \frac{b}{2}n^2 + \left(\frac{b}{2} + d\right)n + c - d - b \Rightarrow O(n^2)$

Arithmetic seq:

$1 + 2 + 3 + \dots + n = \frac{n}{2}(n+1)$

since,

$n + (n-1) + (n-2) + \dots + 2$

$= 2 + \dots + \dots + (n-3) + (n-2) + (n-1) + n$

$= -1 + 1 + 2 + \dots (n-2) + (n-1) + n$

$= \frac{n}{2}(n+1) - 1$

✱ Analyze binary search's TC (recursive)

```
func BinarySearch(L, x, low, high)                                   ——— T(n)
    if (low > high), return (-1)                                     ——— c1
    else mid = (low + high)/2                                        ——— c2
        if (L[mid] = x) return mid.                                  ——— c3
        else
            if (L[mid] > x) return (BinarySearch(L, x, low, mid-1))  — c4 + T(n/2)
            else return (BinarySearch(L, x, mid+1, high))            — c5 + T(n/2)
```

adding up all c cons

✱ Step 1:

$$T(n) = \begin{cases} a + T\left(\frac{n}{2}\right) & n \geq 1 \\ c & n = 0 \end{cases}$$

✱ Step 2:

$$T(n) = T\left(\frac{n}{2}\right) + a$$
$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{2^2}\right) + a$$
$$T\left(\frac{n}{2^2}\right) = T\left(\frac{n}{2^3}\right) + a$$
$$\vdots$$
$$T\left(\frac{n}{2^k}\right) = T\left(\frac{n}{2^{k+1}}\right) + a$$

$\frac{n}{2^k} = 1 \Rightarrow T(1)$

why is $\frac{n}{2^{k+1}} = 0$?

$$\frac{n}{2^{k+1}} = \frac{n}{2^k \cdot 2^1} = \frac{1}{2}$$

but truncate b/c len can't be $\frac{1}{2} \Rightarrow 0$

✱ $n = 2^k$

$$\frac{n}{2^k} = 1$$
$$n = 2^k$$
$$\log_2 n = k$$

$$T(n) = T(0) + (k+1)a$$
$$= c + (k+1)a$$
$$= c + (\log_2 n + 1)a$$
$$= c + \log_2 n a + a$$

$$\Rightarrow T(n) = (c + a) + a\log_2 n \leq B\log_2 n \Rightarrow O(\log_2 n)$$

08/31

Fibonacci:

$$T(n) = \begin{cases} T(n-1) + T(n-2) + a \\ b \end{cases}$$

$F(n) \leq T(n)$

n	1	2	3	4	5	6	7
F_n	0	1	1	2	3	5	8

$F(n) = F_{n-1} + F_{n-2}$

$\geq F_{n-2} + F_{n-2}$

$= 2F_{n-2}$

$T(N) = O(N)$

$N = 1024 + n$

$n = |N| = \log_2 N = 10$

$T(n) = O(\frac{N}{n})$

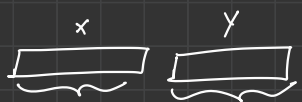
$= O(2^n)$ $\quad N = 2^n$

n = the length of binary string that encodes N

$T(n-1) + T(n-2)$

$x + y$

$F_N \geq \sqrt{2}^N$

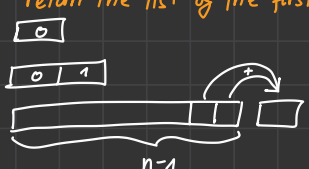


1 2 N

Activity: Design a recursive alg for Fibonacci problem. Twist: instead of returning the Nth Fibonacci number, return the list of the first N Fibonacci numbers.

Base case $\begin{cases} n = 1 \\ n = 2 \end{cases}$

$n \geq 3$



more about Big-O

- Categories of Big O func:
  - logarithmic: $O(\log_2 n)$; $O(\log_2 n)^4$; $O(\log_2(\log_2 n))$
  - polynomial: $O(n)$, $O(\sqrt{n})$; $O(n \log_2 n)$; $O(n^c)$, $c > 0$
  - sub-exponential: $O(2^{(\log_2 n)^k})$, $k > 1$; $O(n^{(\log_2 n)^k})$, $k \geq 1$
  - exponential: $O(2^n)$; $O(2^{cn})$, $c > 1$; $O(2^{n^c})$; $c > 1$, $O(2^{2n})$

Notes:

$\log_2 n^4 \neq (\log_2 n)^4$

$2^{\log_2 n} = n$

$2^{\log_2 n^2} = n^2$ (↳ any polinomal)

But when $2^{(\log_2(n^2))^3} \neq (n^2)^3$ ✗ you CANNOT DO this

$2^{(\log_2 n)^2} = 2^{(\log_2 n)\cdot(\log_2 n)}$

$= 2^{(\log_2 n)^{\log_2 n}} = n^{\log_2 n}$

· Composite rules for Big O:

- $c \times O(f(n)) = O(f(n))$
- $O(f(n)) + O(g(n)) =$ max of both
- $O(f(n)) \times O(g(n)) = f(n) \cdot g(n)$
- $(O(f(n)))^c = O(f(n)^c)$

Proof by math induction

· Remember the recursive def of set $\mathcal{N} = \{1, 2, \dots\}$

create a "bucket" $\mathcal{N}$ to include one number at a time.

$k+1$ is placed in $\mathcal{N}$ if

k is already in $\mathcal{N}$

· recursive def of set S of $sum(k)$'s, $\forall k \in N$:

$sum(k+1) = sum(k) + k$ is placed in S if

$sum(k)$ is already in S

$sum(1) = 1$

$sum(2) = sum(1) + \underbrace{1+1}_{2}$

$\frac{sum(k)}{sum(k+1)} = sum(k) + (k+1)$

· Now suppose for any item $sum(n)$, before it is put in S, it needs to be verified

$$sum(n) = \frac{n}{2}(n+1)$$

$sum(1) = 1$ $\quad \frac{1}{2}(1+1) = \frac{1}{2} \times 2 = 1$

$sum(2) = 1 + 2 = 3$ $\quad \frac{2}{2}(2+1) = 1 \cdot 3 = 3$

$sum(3) = 6$ $\quad \frac{3}{2}(3+1) = 3 \times 2 = 6$

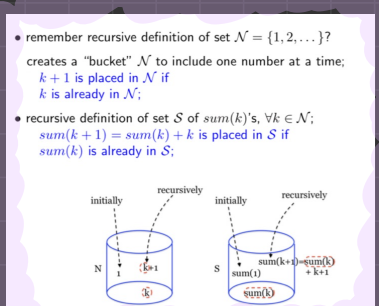
Prove: We want to prove $sum(n) = 1 + 2 + \dots n = \frac{n}{2}(n+1)$

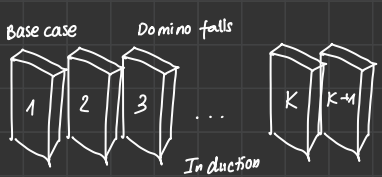
Base case ① $sum(1) = 1 = \frac{1}{2}(1+1)$ proved ✓

Implication { ② assume $sum(k) = \frac{k}{2}(k+1)$

③ induction:

$$\begin{aligned} sum(k+1) &= sum(k) + (k+1) \\ &= \frac{k}{2}(k+1) + (k+1) \\ &= (k+1)\left(\frac{k}{2}+1\right) \\ &= (k+1)\left(\frac{k+2}{2}\right) = \frac{k+1}{2}(k+2) \end{aligned}$$





09/05/23

Prove by induction: that for all $n \geq 0, a \neq 1$

$$1 + a + a^2 + \dots + a^n = \frac{1 - a^{n+1}}{1 - a}$$

let $g(n) = 1 + 2^1 + 2^2 + \dots + 2^n$

$g(n+1) = g(n) + 2^{n+1}$

$P(n) = "g(n) = \frac{1 - a^{n+1}}{1 - a}"$

P(0)

Base step $\quad g(0) = 1 = \frac{1 - a^{0+1}}{1 - a} = \frac{1 - a}{1 - a}$

Assumption assume $g(K) = \frac{1 - a^{K+1}}{1 - a}$

Induction $\quad g(K+1) \xrightarrow{\text{goal}} \frac{1 - a^{K+1+1}}{1 - a}$

via recursive relation $= g(K) + a^{K+1}$

via assumption $= \frac{1 - a^{K+1}}{1 - a} + a^{K+1}$

$= \frac{1 - a^{K+1} + (1 - a) a^{K+1}}{1 - a}$

$= \frac{1 - a^{K+1} + a^{K+1} - a^{K+1+1}}{1 - a}$

$= \boxed{\frac{1 - a^{K+2}}{1 - a}}$ Proven ☆

☆ Exercise: Prove that nth Fibonacci number $F_n \leq 1.8^n$

n	0	1	2	3	4	5	6	7	8
F_n	0	1	1	2	3	5	8	13	21

$$F_n = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ F_{n-1} + F_{n-2} & n \geq 3 \end{cases}$$

$F_n \leq 1.8^n \quad n \geq 1$

Proof:

$n = 1 \quad F_1 = 1 \leq 1.8^1$ ✓

$n = 2 \quad F_2 = 1 \leq 1.8^2$ ✓

assume $F_{K-1} \leq 1.8^{K-1}$, $F_{K-2} \leq 1.8^{K-2}$

Induction: $F_K = F_{K-1} + F_{K-2}$

$\leq 1.8^{K-1} + 1.8^{K-2}$

$\leq 1.8^{K-2} \times 1.8 + 1.8^{K-2}$

$\leq 1.8^{K-2} (1.8 + 1)$ → we want this to be $< 1.8^2$; $1.8 + 1 \to 2.8$

$\leq 1.8^{K-2} \times 1.8^2$ → $1.8^2 \to 3.2$ ✓

$= 1.8^K$ ⇒ Proven

Try these assignments:

① Prove $F_n \leq 1.7^n$, $n \geq 1$

② $1.6^n \leq F_n$

} Try at home to understand

★ What is big-O for T(n), the time func of linear search. Where T(n) was derived as:

$$T(n) = \begin{cases} T(n-1) + a & n > 0 \\ b & n = 0 \end{cases}$$

1. Guess $T(n) = O(n)$
2. We need to prove $\exists c > 0, n_0 > 0$ such that: $T(n) \leq cn$ where $n \geq n_0$ $= P(n)$ → definition of Big O

We can use unfolding to prove

3. Use induction: to prove we have shown that $\exists c = a+b, n_0 = 1$ ✓
   so $T(n) \leq cn$ when $n \geq n_0$

Proof:

① Base case: ~~$n=0$ $T(n) = b \leq c \times 0$~~ why not starting from 0 ?

$n = 1$

$T(n) = T(1-1) + a$

$= T(0) + a$ when $= 1$

$= b + a$

$\leq c \cdot 1$ holds if $c \geq a+b$

② Assume: $T(k-1) \leq c.(k-1)$

③ Induction: $T(k) = T(k-1) + a$

$\leq c(k-1) + a$

$= ck - c + a \leq 0$

$\leq ck$

$c \geq a+b$

$\Rightarrow b \geq a-c$

We can choose $c = a+b$

★ Another example for binary search:

Binary search: $T(n) = \begin{cases} T(\frac{n}{2}) + a & n \geq 1 \\ b & n = 0 \end{cases}$

TC

Prove that $T(n) = O(\log_2 n)$ using induction

Base case: $P(b)$

$P(k) \rightarrow P(k+1)$

Base case: $P(b)$

$P(b) \wedge P(b+1) \wedge \dots \wedge P(k) \rightarrow P(k+1)$

$P(\frac{k+1}{2})$

} Skeleton

Using structural induction: (some case of structural relationships between some running times) to prove:

$T(n) = O(\log_2 n)$

aka this is to prove $\exists c > 0, n_0 \geq 0$ where

$T(n) \leq c \log_2 n$ when $n \geq n_0$

Proof:

Base case: $n = 2$

if an array of 1, cuts in half → $T(0)$

→ choose base $n = 2$ b/c if $n = 1$ → $c \log_2 1 = c \cdot 0 \rightarrow c$ disappear

$T(2) = T(1) + a$

$= T(0) + a + a$ ($T(0) = b$ from original TC)

$= b + a + a$

$= b + 2a$

⇒ For $T(2) = b + 2a \leq c \log_2 2$ to hold $c \geq b + 2a$

Assumption: $T\left(\frac{k}{2}\right) \leq c \cdot \log_2\left(\frac{k}{2}\right)$

Induction: $T(k) = T\left(\frac{k}{2}\right) + a$ (from assumption)

$\leq c \log_2 \frac{k}{2} + a$

$= c(\log_2 k - \log_2 2) + a$

$= c \log_2 k - c + a \leq c \log_2 k$

($c \geq b + 2a$ ⇒ $-c + a$ = negative)

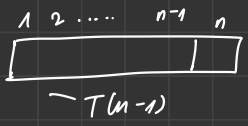
for this to hold, $c \geq a$

goal

⇒ $c = b + 2a$, $n_0 = 2$

☆ Prove insertion sort:

$$T(n) = \begin{cases} an + T(n-1) \\ b \end{cases}$$



Prove that $T(n) = O(n^2)$ using induction

Notes: Stick to definition of TC, find c, n, get into base case

Proof:

$$T(n) = \begin{cases} an + T(n-1) & n \geq 1 \\ b & n = 0 \end{cases} \Rightarrow T(n) = O(n^2) \text{ for insertion sort.}$$

This is to prove $\exists c > 0, n_0 \geq 0$ where $\boxed{T(n) \leq cn^2, n \geq n_0}$

① Base case: $n = 1$

$T(1) = a + T(0)$

$= a + b$

For $T(1) = a + b \leq cn^2 = c$ to hold $\boxed{c \geq a + b}$

② Assumption:

$T(k-1) \leq c(k-1)^2$

③ Induction:

$T(k) = ak + T(k-1)$

$\leq ak + c(k-1)^2$

$\leq ak + c(k^2 - 2k + 1)$

$\leq ak + ck^2 - 2ck + c$

$\leq ck + ck^2 - 2ck + c$

$\leq ck^2 - ck + c$

$= ck^2 - (ck - c)$

$\leq ck^2$

goal: get $T(k) \leq ck^2$

because $c \geq a + b \Rightarrow a \leq c$

since $k \geq 1$, $ck - c$ is always ≥ 0 $\Rightarrow -(ck - c) \leq 0$

We have proven $T(n) = O(n^2)$ for $c = a + b, n_0 = 1$

Quiz Review - Old Exam

① Let function $T(n) = 1$ when $n = 1$ and $T(n) = T(n-1) + n$ for $n \geq 2$. Use induction to prove $T(n) = O(n^2)$

This is to prove $\exists c$, and k such that $T(n) \leq cn^2$ when $n \geq k$.

Proof:

★ Base case: $n = 1 = k$

We have $T(n) = T(1) = 1$, which can be satisfied with $c \geq 1$

★ Assumption: $T(n-1) \leq c(n-1)^2$

★ Induction:

goal: trying to get $T(n) \leq cn^2 + \dots$

$$T(n) = T(n-1) + n$$
$$\Rightarrow T(n) \leq c(n-1)^2 + n$$
$$\leq c(n^2 - 2n + 1) + n = cn^2 - 2cn + c + n$$
$$\leq cn^2 - 2cn + c + cn \quad \text{holds when } c \geq 1$$
$$\leq cn^2 - (2cn - c - cn)$$
$$\leq cn^2 - (cn - c)$$
$$\leq cn^2 \quad \text{because } c \geq 1, n \geq 2$$

$\Rightarrow$ By choosing $c \geq 1$, we have $T(n) \leq cn^2$. Thus, we have $c \geq 1$ and $k \geq 1$.

② Prove by induction, $F_n \leq 1.7^n$ for all $n \geq 0$.

n	0	1	2	3	4	5	6	7	8
F_n	0	1	1	2	3	5	8	13	21

Proof:

★ Base case: $n = 0$

$F(0) = 0 \leq 1 = 1.7^0$ ✓

$F(1) = 1 \leq 1.7^1 = 1.7$ ✓

★ Assumption: for $k \geq 1$

assume $F_{k-1} \leq 1.7^{k-1}$

$F_{k-2} \leq 1.7^{k-2}$

★ Induction:

$$F_k = F_{k-1} + F_{k-2}$$
$$F_k \leq 1.7^{k-1} + 1.7^{k-2}$$
$$\leq 1.7^{k-2} \cdot 1.7 + 1.7^{k-2}$$
$$\leq 1.7^{k-2}(1.7 + 1) \to 2.7$$

$$F_n \leq 1.7^{k-2} \times 1.7^2 \quad (1.7^2 \to 2.89)$$
$$F_n \leq 1.7^{k-2+2}$$
$$\Rightarrow \boxed{F_n \leq 1.7^k}$$

so $2.7 < 2.89$

So we have proven that $F_n \leq 1.7^n$ for all $n \geq 0$ by induction

③ Prove $1.6^n \leq F_n$

Proof:

★ Base case: $n = 20$ (holds when $n \geq 20$)

$1.6^{20} \leq F_{20}$ ✓

★ Assumption: For $n \geq 21$

Assume $1.6^{k-1} \leq F_{k-1}$

$1.6^{k-2} \leq F_{k-2}$

★ Induction:

$F_k = F_{k-1} + F_{k-2}$

$F_k \geq 1.6^{k-1} + 1.6^{k-2}$

$\geq 1.6^{k-2} \times 1.6 + 1.6^{k-2}$

$\geq 1.6^{k-2}(1 + 1.6) = 1.6^{k-2} \times 2.6$

$\geq 1.6^{k-2} \times 1.6^2$ (b/c $1.6^2 < 2.6$)

$\geq 1.6^{k-2+2}$

$\geq \boxed{1.6^k}$ → Proven !

⇒ We have proven that $1.6^n \leq F_n$ for $\boxed{n \geq 20}$

Draft / Thought process:

$1.6^{k-2} \times 2.6$

$1.6^2 = 2.56$

Redo some induction proof for recursion. REWORK

Skip this - it's just repeated work for practice

★ What is big-O for T(n), the time func g linear search. Where T(n) was derived as:

$$T(n) = \begin{cases} T(n-1) + a & n > 0 \\ b & n = 0 \end{cases}$$

① Guess: $T(n) = O(n)$

② Goal: Use induction to prove $\exists c, k$ so that $T(n) \leq cn$ where $n \geq k$

★ Base case: $n = 1$

$T(1) = T(0) + a = b + a \leq c \cdot 1$ holds if $\boxed{c \geq a + b}$ ✓

★ Assumption:

$T(k-1) \leq c(k-1)$

★ Induction:

$T(k) = T(k-1) + a$

$\Rightarrow T(k) \leq c(k-1) + a$

$= ck - c + a$

$\boxed{1 \leq ck}$ negative (< 0) ⇒ b/c $c \geq a + b$ ⇒

Have proven that $T(n) = O(n)$ when $c = a + b$, $n_0 = 1$

④ Binary search TC: $T(n) = \begin{cases} T(\frac{n}{2}) + a & n \geq 1 \\ b & n = 0 \end{cases}$

Prove that $T(n) = O(\log_2 n)$ using induction

★ For TC analysis of recursion: Use induction, only use the recurrence relation for base case. Assumption + Induction, use on $T(k) \leq ck$

1. Guess $T(n) = O(\log n)$

2. Goal: use induction to prove $\exists c$ and k such that $T(n) \leq c \log_2 n$ where $n \geq k$.

Proof:

★ Base case: $n = 2$ (b/c $n = 1$ doesn't work)

$T(2) = T(1) + a$

$= T(0) + a + a$

$= b + 2a$

For $T(2) = b + 2a \leq c \log_2 n$

$= b + 2a \leq c \log_2 1 = c$, holds when $c \geq b + 2a$

★ Assumption: $T(\frac{k}{2}) \leq c \log_2 (\frac{k}{2})$

★ Induction: $T(k) = T(\frac{k}{2}) + a$

$\leq c \log_2 (\frac{k}{2}) + a$

$\leq c(\log_2 k - \log_2 2) + a = c \log_2 k - c + a$

$-c + a \leq 0$

$\leq c \log_2 k$

$\Rightarrow$ We have proven $T(n) = O(\log_2 n)$ for $c = b + 2a$, $n_0 = 2$

⑤ Analyze TC for recursion

```
function LinearSearch(L, x, n);                    <---- T(n)
if (n = 0) return ("not found");                   <----- c1
else
  if (L[n]=x) return (n);                          <----- c2
  else return (LinearSearch(L, x, n-1));           <----- c3 + T(n-1)
```

$T(n) = c_1$ or c_2 or $c_3 + T(n-1)$

$T(n) = c_3 + T(n-1)$

$T(0) = c_1$ ⟶ Base case

Step 1:

$T(n) = \begin{cases} c_3 + T(n-1) & n = 1 \\ c_1 & n = 0 \end{cases}$

Step 2:

$T(n) = T(n-1) + c_3$

$T(n-1) = T(n-2) + c_3$

$T(n-2) = T(n-3) + c_3$

$\vdots$

$T(1) = T(0) + c_3$

$\Rightarrow T(n) = T(0) + c_3 \times n$

$= c_1 + c_3 n$

Step 2: b/c $c_1 + c_3 n \leq (c_1 + c_3) n$ (c, $n_0 = 1$) $\Rightarrow O(n)$

⑥ ★ Analyze worst case time complexity for recursive selection sort algorithm:

```
S(n) _ func FindMax(L,n)
   b _ if n>1
     S(n-1) FindMax(L,n-1)
       c' if (L[n-1] > L[n])
       c'' swap(L[n-1], L[n])
```

```
T(n) _ func SelectionSort(L,n)
        if n>1 _ c
   S(n) ____ FindMax(L,n)
T(n-1) ____ SelectionSort(L,n-1)
```

Step 1: $S(n) = S(n-1) + c + c' + c'' = S(n-1) + c$ (c)

$\Rightarrow S(n) = \begin{cases} S(n-1) + b & \text{when } n>1 \\ c & n=1 \end{cases}$

$T(n) = S(n) + T(n-1) + c$

$T(n) = a + bn + T(n-1) + c$

$T(n) = T(n-1) + bn + d$

$\Rightarrow T(n) = \begin{cases} T(n-1) + bn + d & \text{when } n>1 \\ c & n=1 \end{cases}$

$S(n) = S(n-1) + b$

$S(n-1) = S(n-2) + b$

$\vdots$

$S(2) = S(1) + b$

$\Rightarrow S(n) = S(1) + b \times (n-1)$

$S(n) = S(1) - b + bn$

$S(n) = \boxed{a + bn}$

Step 2: $T(n) = T(n-1) + bn + d$

$+ \begin{cases} T(n-1) = T(n-2) + b(n-1) + d \\ T(n-2) = T(n-3) + b(n-2) + d \\ \vdots \\ T(2) = T(1) + b2 + d \end{cases}$

$T(n) = T(1) + b[n + (n-1) + (n-2) + \dots + 2] + (n-1)d$

$T(n) = T(1) + b\left[\frac{n}{2}(n+1) - 1\right] + (n-1)d$

$\Rightarrow T(n) = c + \frac{b}{2}n^2 + \frac{b}{2}n - b + nd - d$

$\Rightarrow T(n) = \frac{b}{2}n^2 + \left(\frac{b}{2} + d\right)n + \underbrace{c - b - d}_{c}$ ($n_0 = 1$)

$\Rightarrow T(n) = \frac{b}{2}n^2 + \left(\frac{b}{2} + d\right)n + a \leq \left(\frac{b}{2} + \frac{b}{2} + d + a\right)n^2$

$\Rightarrow T(n) = \boxed{O(n^2)}$

⑦ ★ Analyze binary search's TC (recursive)

```
func BinarySearch(L, x, low, high) ____ T(n)
    if (low > high), return (-1) ____ c1
    else mid = (low + high)/2 ____ c2
        if (L[mid] = x) return mid. ____ c3
        else
            if (L[mid] > x) return (BinarySearch(L, x, low, mid-1)) ____ c4 + T(n/2)
            else return (BinarySearch(L, x, mid+1, high)) ____ c5 + T(n/2)
```

all c's

Step 1: $T(n) = \begin{cases} a + T\left(\frac{n}{2}\right) & n \geq 1 \\ c & n = 0 \end{cases}$

Step 2:

$+ \begin{cases} T(n) = T\left(\frac{n}{2}\right) + a \\ T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + a \\ T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + a \\ \vdots \\ T\left(\frac{n}{2^k}\right) = T\left(\frac{n}{2^{k+1}}\right) + a \end{cases}$ (len(0) of list)

$T(n) = T(0) + a \cdot (k+1)$

$\frac{n}{2^k} = 1$

$\log_2 n = k$

$= c + (k+1)a = c + (\log_2 n + 1)a = c + \log_2 n\, a + a \Rightarrow T(n) = (c+a) + a\log_2 n \leq B\log_2 n$

$O(\log_2 n)$

09/07/23

What we've covered:

- TC of iterative algorithms - $T(n)$
- Big-O
- TC of recursive algorithm - $T(n)$
- Mathematical induction

Recursive formula:

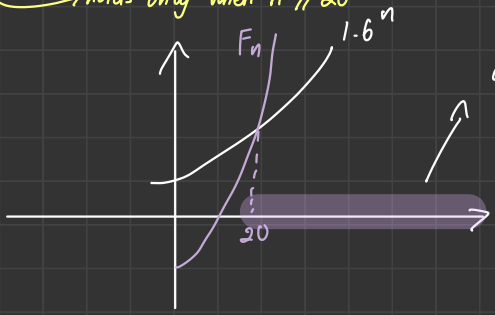
$$F_n = \begin{cases} 0 & n = 1 \\ 1 & n = 2 \\ F_{n-1} + F_{n-2} & n \geq 3 \end{cases}$$

Back to exercise:

$1.6^n \leq F_n$ holds for all $n \geq 1$?

$1.6^{20} \leq F_{20}$ → holds → base case: 20 → $n \geq 20$

→ holds only when $n \geq 20$



only holds

⇒ can choose any base case that works, doesn't need to stick to the original formula.

Quiz 2 Key

(1)

1. (**10 points**) Consider the following recursive algorithm `DoSomething`. Assume that the algorithm has the worst case time $T(n)$ on list L of n elements.

```
function DoSomething(L, n); // L is a list indexed from 1 to n, n>=1
  if (n = 1)
     return (L[1]);                          — a
  else
    if (L[n] > L[n/2])
       swap(L[n/2], L[n]);  // assumed n can be evenly divided by 2 — b
    return (DoSomething(L, n-1)); T(n-1)
```

a) Formulate $T(n)$ as recursive func, including the base case:

$$T(n) = \begin{cases} a & \text{base case when } n = 1 \\ T(n-1) + b & \text{when } n \geq 2 \end{cases}$$

b) To prove $T(n) = O(n)$, its equivalent to proving that $\exists c > 0, n_0 > 0$ such that $T(n) \leq cn$ when $n \geq n_0$

(2) (1) To prove $F_n \leq 1.8^n$ where F_n is the n^{th} Fibonacci number

=) Base cases should be : $F_1 \leq 1.8$

$F_2 \leq 1.8^2$

(2)

(2) Let $T(n)$ be a recursive function defined with base case $n = 2$. To prove certain statement $\mathcal{P}(n)$ about $T(n)$ with a proof-by-induction, we need to consider the base case in the proof -by-induction to be $n = b$, where b is a constant that satisfies [C] only [4] of the following statements: each additional answer will result in 1 point deduction

[A] $b = 2$ so to be consistent with the base case definition for $T(n)$;

[B] b has to the smallest number ≥ 2 and allows statement $\mathcal{P}(b)$ to hold for $T(b)$;

[C] b can be any constant number ≥ 2 that allows statement $\mathcal{P}(b)$ to hold for $T(b)$;

[D] b can be either < 2, $= 2$, or > 2, since all we need is a constant, as long as it allows statement $\mathcal{P}(b)$ to hold for $T(b)$;

[E] None of above, because the base case for $T(n)$ is completely different from the base case for proof-by-induction;

(3) Let function $sum(n) = 1 + 2 + 3 + \ldots + n$ where $n \geq 1$. To prove statement $sum(n) = \frac{n}{2}(n+1)$ by induction, at the inductive step $n = K$, $K \geq 2$

(1) We need to use the following assumption :

$$sum(K-1) = \frac{K-1}{2}(K-1+1) = \frac{K-1}{2} \cdot K$$

$$sum(K) = sum(K-1) + K$$

(2) The goal of inductive step is to show $sum(K) = \frac{K}{2}(K+1)$

(4)

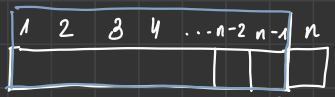
4. (**10 points**) Complete the following pseudo-code for `InsertionSort` on input list L of n elements, where subroutine function `Insert`(L, n) inserts the element from position n into the sorted prefix-sublist $L[1..n-1]$.

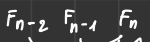
```
function InsertionSort(L, n);    // Sort list L in non-decreasing order;
                                 // L is indexed from 1 to n, n ≥ 1;
  if n > 1   [1]
    InsertionSort(L, n − 1);   [2]
    Insert(L, n);              [2]


function Insert(L, n);
  if n > 1   [1]
     if L[n − 1] > L[n]   [2]
        Swap(L[n − 1], L[n]);   [2]
        Insert(L, n − 1);
```

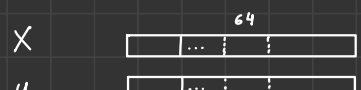
elementary algorithms

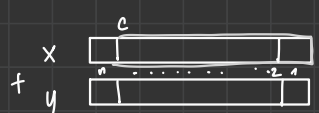
DIVIDE & CONQUER

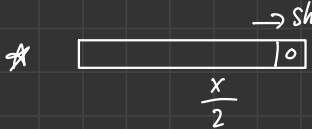




★ Suppose you have 2 long numbers x and y

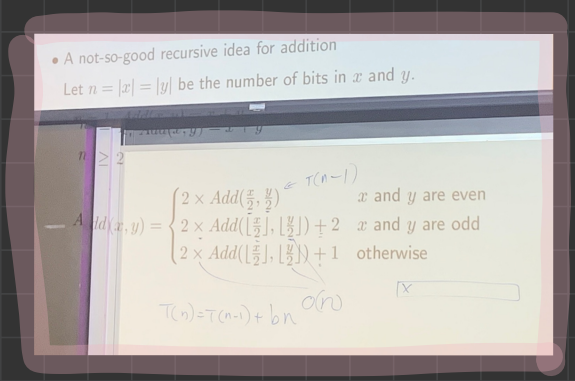







```
FibSeq(n):                    T(n)
if n=1  return (0)
if n=2  return (0)·(1)
else:
    A = FibSeq(n-1);          T(n-1)
    return (A · (A(n-1) + A(n-2)))
```

take advantage of pattern & come up w/ a recursive alg.



(1) Power of divide and conquer

- Multiplication of 2 n-bits numbers:
  + Multiplication as repeat additions: $x \times y = \underbrace{y + y + y + \ldots + y}_{x \text{ times}}$

—— $T(n) = O(n)$

$|x| = |y| = n$

$O(n) \times X$ $\quad |X| = n$

$x \leq 2^n$ ← the length can be as big as n

a diff strategy: recursive

or you can: $x \to \frac{x}{2}$

$x \times y = \boxed{y + y + y + \ldots + y}$

↑ do double ↑ no need to compute this ⇒ save resources

$\Rightarrow \boxed{x \cdot y}$

★ $Multi(x,y) = \begin{cases} 2 \times Mult(x, \frac{y}{2}) & \text{if } y \text{ even} \\ x + 2 \times Mult(x, \lfloor \frac{y}{2} \rfloor) & \text{if } y \text{ odd} \end{cases}$

★ $T(n)$ for $Mult(x,y)$ where $|y| = n$:

$$T(n) = \begin{cases} an & n = 1 \\ T(n-1) + bn & n > 1 \end{cases}$$

length of x

⇒ quadratic

✱ Induction example:

Claim: $T(n) = O(n^2)$. That is, $\exists c > 0, n_0 > 0$ such that $T(n) \leq cn^2$ where $n \geq n_0$. (∀t)

Prove by math induction:)

choosing $c = \max\{a, b\}$ satisfies the proof !!!

① Base case: $n = 1$

$T(n) = T(1) = a \cdot 1 \overset{?}{\leq} c \cdot 1$ holds when $c \geq a$

② Assumption: $n = k$

$T(k) \leq ck^2$ (by assumption)

③ Induction: $n = k+1$

$T(k+1) = T(k) + b(k+1)$

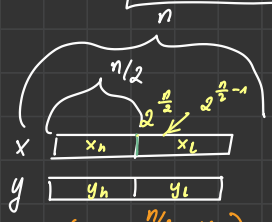
$\leq ck^2 + b(k+1)$

$\leq ck^2 + bk + b$

$\leq c(k+1)^2 = ck^2 + 2ck + c$ holds if choose $c \geq b$

✱ Back to divide & conquer

- Multiplication:



$$\Rightarrow x \cdot y = \left(x_h \cdot 2^{n/2} + x_l\right) \times \left(y_h \times 2^{\frac{n}{2}} + y_l\right)$$

$$= x_h y_h 2^n + \left(x_h y_l + x_l y_h\right) 2^{\frac{n}{2}} + x_l y_l$$

$$T(n) = \begin{cases} a & n = 1 \\ 4T\left(\frac{n}{2}\right) + bn & n \geq 2 \end{cases}$$

rough draft

$4^{\log_2 n} = 2^{\log_2 n^2}$

$4 \times 4 \times 4 \times 4 \ldots \times 4$ ($\log_2 n$ times) ← cut it half

$4^{\log_2 n} = 2^{(2\log_2 n)} = 2^{\log_2 n^2}$

09/12/23

Multiplication:

$$T(n) = \begin{cases} a & n=1 \\ 4T\left(\frac{n}{2}\right) + 6n & n \geq 2 \end{cases}$$

to make induction easier to prove

Prove $T(n) = O(n^2)$ is equivalent to proving $\exists c > 0,\ n_0 \geq 0$ s.t. $T(n) \leq cn^2 + xn$ when $n \geq n_0$.

Prove by induction:

★ Base: $n = 1$

For $T(1) = a \leq c \cdot 1 \Rightarrow \boxed{c \geq a}$

★ Assumption: $n = \frac{k}{2}$ $\quad T\left(\frac{k}{2}\right) \leq c\left(\frac{k}{2}\right)^2 + x\frac{k}{2}$

Induction: $n = k$

$$\begin{aligned} T(k) &= 4T\left(\frac{k}{2}\right) + bk \\ &\leq 4\left(c\left(\frac{k}{2}\right)^2 + x\frac{k}{2}\right) + bk \\ &= ck^2 + \boxed{2xk + bk} \\ &\leq ck^2 + \boxed{xk} \quad \checkmark \end{aligned}$$

$2xk + bk \leq xk$

$x + b \leq 0 \longrightarrow ?$

$x \leq -b$

- Multiplication (a better solution)

$$\begin{aligned} xy &= (x_h 2^{\frac{n}{2}} + x_\ell)(y_h 2^{\frac{n}{2}} + y_\ell) \\ &= x_h y_h 2^n + (x_h y_\ell + x_\ell y_h) 2^{\frac{n}{2}} + x_\ell y_\ell \end{aligned}$$

where with ℓ "x" operation:

$$(x_h y_\ell + x_\ell y_h) = (x_h + x_\ell)(y_h + y_\ell) - x_h y_h - x_\ell y_\ell$$

Exercise: slide

matrix manipulation

- For 2×2 matrix: $A_{(2\times2)} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

$A_{(2\times2)} \times B_{(2\times2)} = C_{(2\times2)}$

coventional way: 8 scalar multiplication needed

- For $n \times n$ matrices: $A_{n\times n} \times B_{n\times n} = C_{n\times n}$

$$A_{n\times n} = \begin{bmatrix} X_{(\frac{n}{2}\times\frac{n}{2})} & Y_{(\frac{n}{2}\times\frac{n}{2})} \\ Z_{(\frac{n}{2}\times\frac{n}{2})} & W_{(\frac{n}{2}\times\frac{n}{2})} \end{bmatrix}$$

→ coventional way...

⇒ Better solution:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} ax + bz & ay + bw \\ cx + dz & cy + dw \end{bmatrix}$$

more clever w/ algebra:

$$\begin{cases} s_1 = a(y-w) & s_5 = (a+d)(x+w) \\ s_2 = (a+b)w & s_6 = (b-d)(z+w) \\ s_3 = (c+d)w & s_7 = (a-c)(x+y) \\ s_4 = d(z-w) \end{cases}$$

!!!

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} s_4 + s_5 + s_6 - s_2 & s_1 + s_2 \\ s_3 + s_4 & s_1 + s_5 - s_3 - s_7 \end{bmatrix} \Rightarrow$$

7 multiplications + 18 additions/substractions

Example: merge sort:

Input: a list $L[1 \dots n]$ of n elements

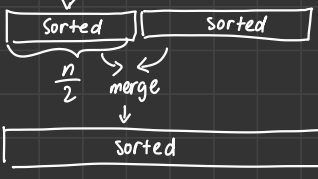
Output: list $L[1 \dots n]$, a permutation of L such that

$$\forall i, 1 \leq i < n, L[i] \leq L[i+1]$$

$T(n) = T(n-1) + O(n)$

Selection, Insertion } $O(n^2)$

$\frac{n}{2}$ → $\frac{n}{2}$



⇒ $T(n) = \begin{cases} a & n = 1 \\ 2T(\frac{n}{2}) + bn & n \geq 2 \end{cases}$

★ Solution:

$T(n)$ — MergeSort (low, high, L)

n = (high - low + 1)

if (low < high)

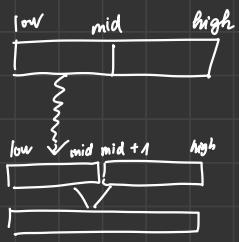
mid = (low + high)/2 — c

MergeSort (low, mid, L) — $T(\frac{n}{2})$

MergeSort (mid + 1, high, L) — $T(\frac{n}{2})$

n — Merge (low, mid, high, L) → // merging sorted sublists $L[low, mid]$ and $L[mid+1, high]$

↓ n because



Prove: ★ Use unfolding

Claim $T(n) = O(n \log_2 n)$

$T(n) = 2T(\frac{n}{2}) + bn$

$T(\frac{n}{2}) = 2T(\frac{n}{2^2}) + b(\frac{n}{2})$

$T(\frac{n}{2^2}) = 2T(\frac{n}{2^3}) + b\frac{n}{2^2}$

$\vdots \quad \vdots \quad \vdots$

$T(2) = 2T(1) + b \cdot 2$

08/14/23

Continue proving:

$$T(n) = \begin{cases} 4T\left(\frac{n}{2}\right) + bn & n \geq 2 \\ a & n = 1 \end{cases}$$ ⇒ multiplication

$T(n) \leq cn^2 - xn$

Task: to prove that $\exists c, n_0 > 0$ st $T(n) \leq cn^{1.6}$, $n \geq n_0$

$n = 1$ $\quad T(1) \leq c \times 1^{1.6}$

assume $n = \frac{k}{2}$ $\quad T\left(\frac{k}{2}\right) \leq c\left(\frac{k}{2}\right)^{1.6}$

★ Look more into this!

Goal $T(k) \leq ck^{1.6}$

$$= 3 \times T\left(\frac{k}{2}\right) + bk$$
$$\leq 3 \times c\left(\frac{k}{2}\right)^{1.6} + bk$$
$$= c + \frac{3}{2^{1.6}} k^{1.6} + bk$$
$$\Downarrow$$
$$\leq ck^{1.6}$$
$$\leq 0.99\, ck^{1.6} + bk$$
$$\leq (1 - 0.01)ck^{1.6} + bk$$
$$= ck^{1.6} \underbrace{(-0.01ck^{1.6} + bk)}_{\leq 0} \quad 0.01ck^{1.6} \geq bk \quad c \geq 100b$$
$$\leq ck^{1.6}$$

★ No quiz monday

recursively sort 2 halves and merge them

Merge sort

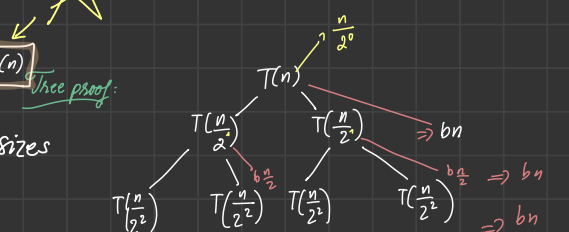
n = high − low

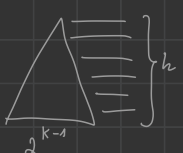
base case: $T(1) = c$

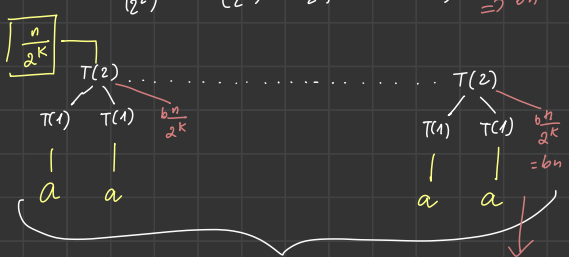
recursive case: $T(n) = 2T\left(\frac{n}{2}\right) + T_{\text{merge two}}(n)$

- Mergesort idea:
  - partition L into 2 sublists of equal sizes
  - sort the 2 sublists
  - merge the sorted sublists into one

Three psoof:







$\frac{n}{2^k} = 2$

$n = 2^{k+1}$

$k + 1 = \log_2 n$

$$= a \times 2^{k+1} + b \cdot n \times (k+1)$$
$$= a \cdot n + b \cdot n \cdot \log_2 n$$
$$= \boxed{(a+b)\, n \log_2 n}$$

☆ Prove that merge sort TC $T(n) = O(n \log_2 n)$

Prove w/ unfolding:

$$T(n) = \begin{cases} a & n = 1 \\ 2T(\frac{n}{2}) + bn & n \gg 2 \end{cases}$$

$$T(n) = 2T(\frac{n}{2}) + bn$$

$$2T(\frac{n}{2}) = 2^2 T(\frac{n}{2^2}) + 2b\frac{n}{2}$$

$$2^2 T(\frac{n}{2^2}) = 2^3 T(\frac{n}{2^3}) + 2^2 b\frac{n}{2^2}$$

$$\vdots \quad \vdots \quad \vdots$$

$$+ \quad 2^k T(\frac{n}{2^k}) = 2^{k+1} T(\frac{n}{2^{k+1}}) + 2^k b\frac{n}{2^k}$$

$$T(n) = 2^{k+1} T(\frac{n}{2^{k+1}}) + bn(k+1)$$

where $\frac{n}{2^{k+1}} = 1 \Rightarrow k+1 = \log_2 n$

$$\Rightarrow T(n) = nT(1) + bn\log_2 n = O(n\log_2 n)$$

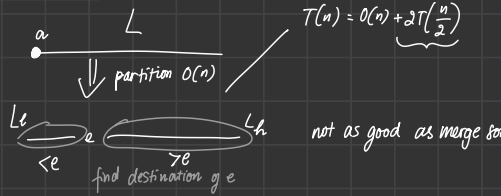
Prove w/ induction:

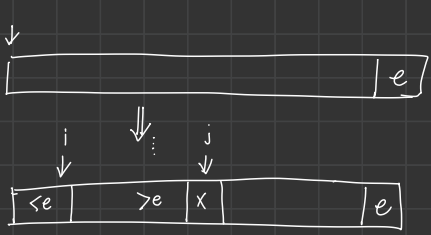
⇒ Prove $\exists c > 0, n_0 > 0$ s.t $T(n) \leq cn\log_2 n$ where $n \geq n_0$

- Base case: $n = 1 \Rightarrow T(1) \leq c \times 1 \times \log_2 1$?
- Assumption: when $n = \frac{k}{2}$, $T(\frac{k}{2}) \leq c\frac{k}{2}\log_2(\frac{k}{2})$
- Induction: when $n = k$

$$T(k) = 2T(\frac{k}{2}) + bk \quad \text{by recursive formula}$$
$$\leq 2c\frac{k}{2}\log_2(\frac{k}{2}) + bk \quad \text{(from assumption)}$$
$$= ck\log_2\frac{k}{2} + bk$$
$$= ck(\log_2 k - \log_2 2) + bk$$
$$= ck\log_2 k - ck + bk$$
$$\leq ck\log_2 k \quad \text{— if choose } c > b$$

quick sort:





x compared to e

if x > e j = j+1

else {x < e} swap(y, x)

j = j+1

i = i+1

Idea of quick sort:

- select a pivot elem e from input list L
- partition list L into 2 sublists L_h & L_l s.t:

$$\forall x \in L_h, x > e$$
$$\forall x \in L_l, x \leq e$$

- recursively sort the 2 sublists L_h and L_l separately

☆ function quicksort (L, low, high):

```
if (low < high)
    k = partition (L, low, high)
    quicksort (L, low, k-1)
    quicksort (L, k+1, high)
return L;
```

```
function partition (L, p, r)
    e = L[r]
    i = p-1
    for j = p to r-1
        if L[j] ≤ e
            i = i+1
            exchange (L[i], L[j])
    exchange (L[i+1], L[r])
    return i+1
```

- choose a pivot
- initialize left & right pointers
- compare w/ pivot & move pointers to partition elems
- swap when needed to arrange elements
- place pivots in final position

Quicksort time complexity:

- worst case: $T(n) = T(n-1) + bn \Rightarrow \boxed{O(n^2)}$
- avg. case (idea): $T(n) = 2T\left(\frac{n}{2}\right) + bn \Rightarrow \boxed{O(n \log_2 n)}$ → high probability for with randomized technique

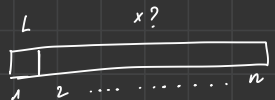
09/18/23

Prove merge sort = $T(n) = O(n \log_2 n)$ w/ induction.

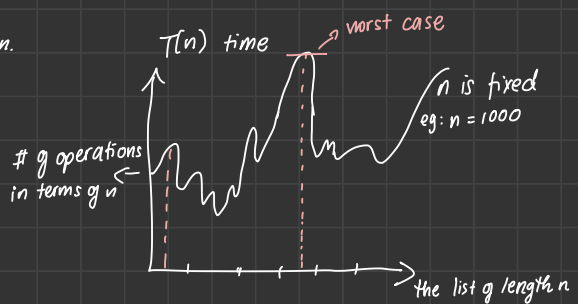
★ Quick sort: 3 7 6 2 9 4 | 5 ← pivot

3 2 4 5 7 6 9

ex

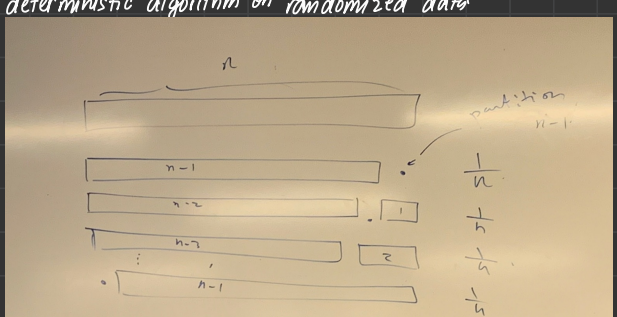


$1 + 2 + \ldots + n = \frac{n}{2}(n+1)$



★ Prove averaged time complexity:

- deterministic algorithm on randomized data

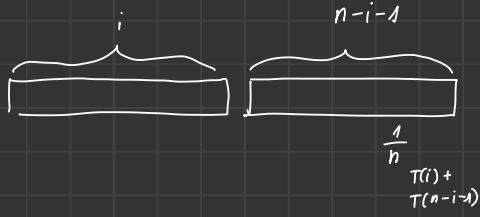


$$\tilde{T}(n) = (n-1) + \frac{1}{n}\sum_{j=0}^{n-1}\left[\tilde{T}(i) + \tilde{T}(n-j-1)\right] = (n-1) + \frac{2}{n}\sum_{i=0}^{n-1}\tilde{T}(i) \Rightarrow T(n) = O(n \log n)$$

09/19/23

Prove random quick sort algorithm is $O(n \log_2 n)$

$|L_l| = i$ $|L_h| = n-i-1$



random quick sort:
- randomly pick up a pivot
- partition the list into L_l, L_h
- recursively call
- read quick sort on L_l
- ..... L_h

$$T(n) = \sum_{i=0}^{n-1} \frac{1}{n} \underbrace{\left[(n-1) + T(i) + T(n-i-1)\right]}_{\text{the amount of time when pivot partition list into } L_l, L_h}$$

$|L_l| = i, |L_h| = n-i-1$

$$= (n-1) + \frac{1}{n} \sum_{i=1}^{n-1} \left(T(i) + T(n-i-1)\right)$$

$$= (n-1) + \frac{2}{n} \sum_{i=0}^{n-1} T(i)$$

Try plugging numbers:

?

$n = 4$ $i = 4, 1, 2, 3$

$n \geq 2 \rightarrow$ Base case

$n = 1$ $T(n) = a$

More proof is in slides

$$T(n) = T(i) + T(n-i-1) + \underbrace{\Theta(n)}_{\text{time spent on partitioning}}$$

- $T(i)$: TC for the left subarray with i elems
- $T(n-i-1)$: TC for right subarray with $(n-i-1)$ elems (not counting pivot)

$$1 \times \log_1 + 2 \times \log 2 + \ldots + (k-1)\log(k-1)$$

$$\sum_{i=0}^{k-1} i = 0 + 1 + 2 + \ldots + k-1$$

$$= \frac{k-1}{2}(k-1+1)$$

$$= \frac{k(k-1)}{2}$$

What algorithms have the following recurrences for running time:

$T(n) = T(n-1) + b$	Linear search
$T(n) = T(n-1) + bn$	Insertion sort / Selection sort
$T(n) = T(\frac{n}{2}) + b$	Binary search
$T(n) = dT(\frac{n}{2}) + bn, d = 2, 3, 4$	Multiplication
$T(n) = dT(\frac{n}{2}) + bn^2, d = 7, 8$	Matrix
$T(n) = T(\alpha n) + T(\beta n) + bn$ for $\alpha + \beta = 1$	Balanced quick sort
$T(n) = T(\alpha n) + T(\beta n) + T(\gamma n) + bn$ for $\alpha + \beta + \gamma = 1$	Random quicksort

Selection: deterministic & randomized

Problem selection:

Input: a list L and rank k

Output: the k^{th} smallest element in L

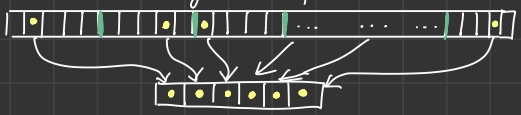
(median = $\frac{k^{th}}{2}$)

deterministic: worst case $O(n)$

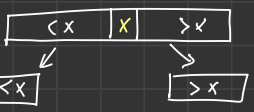
randomized: average d case $O(n)$

3.1 Deterministic SELECTION algorithm: (Blum, Floyd, Pratt)

- For now, assume n = multiple of 5
- For each block of size 5: find 3rd smallest value $\approx$ representative — $\frac{n}{5} \cdot O(1) = O(n)$



- Copy representatives to new list — $O(n)$
- Find median of representatives $\rightarrow x$ — $T(\frac{n}{5})$ (How? use the alg (recursive))
- Use x as a pivot, partition input — $O(n)$



- Recurse left / right if necessary — $T(?)$

★ Deterministic selection: idea of linear time deterministic selection (A, n, k) algorithm.

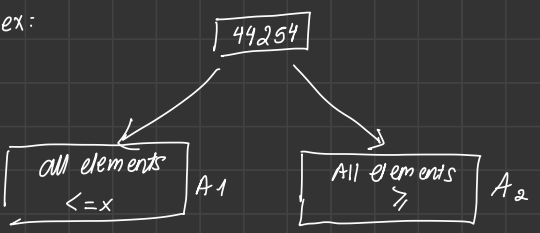
- Numbers in A are grouped into $\frac{n}{5}$ groups, w/ 5 numbers in each group
- The 3rd largest number in every group is chosen, all such numbers are placed in new set M.

ex: $\frac{n}{5} = 40$ $\quad |M| = \frac{n}{5} = 40$

$$\text{median} = 20^{th} = \left(\frac{40}{2}\right)^{th}$$

$$= \left(\frac{\frac{n}{5}}{2}\right)^{th} = \left(\frac{n}{10}\right)^{th}$$

ex:



$$\text{median} = \frac{\frac{M}{2}}{2} = \frac{\frac{N}{5}}{2} = \frac{N}{10}$$

★ Summary of algorithm:

- Input: list A of n elems, and parameter k. (goal: to find k^{th} smallest elem from A)
- group elements A into groups of 5, resulting in $\frac{n}{5}$ groups.
- for each group, pick the third largest elem. put such elems in all group in M.
- let x = selection $(M, \frac{n}{5}, \frac{n}{10})$, a recursive call to selection; x is the $(\frac{n}{10})^{th}$ smallest elem in M, or median in M
- use x as pivot to partition A into A_1 and A_2 such that

  $\forall y \in A_1, y \leq x$, and $\forall z \in A_2, x < z$.

find $(\frac{n}{10})^{th}$ smallest number in M

★ Continue:

09/21/23

linear time deterministic selection (A, n, k) algorithm:

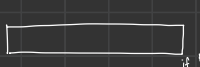
- r = rank (x) in A
- if $r = k$ then alg returns x

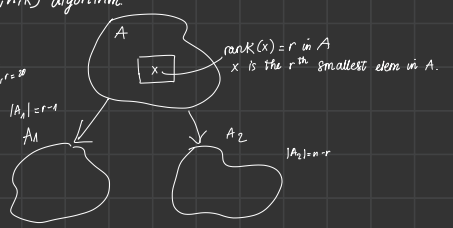
otherwise,

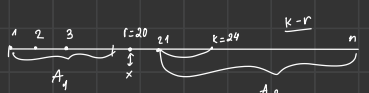
if $k < r$, return selection $(A_1, r-1, k)$

else, return selection $(A_2, n-r, k-r)$

Recursion







Algorithm in slides

Algorithm select(A, n, k) ——— $T(n)$

1. if $n < 140$, simply sort A and return $A[k]$ ——— c_1
2. Find a pivot
3. group elems in A into group of 5 elems ——— $\leq c_3 n$
4. place 3rd largest elems from all groups into list M ——— $\leq c_4 n$
5. $x =$ SELECT$(M, \frac{n}{5}, \frac{n}{10})$ ——— $T(\frac{n}{5})$
6. let $r =$ rank(x) in A ——— $\leq c_6 n$
7. if $k = r$, return $A[r]$ ——— c_7
8. partion A into $A_1 = \{y : y \leq x\}$, $A_2 = \{z : x < z\}$ ——— $\leq c_8 n$
9. if $k < r$, return select$(A_1, r-1, k)$ ——— $T(|A_1|)$
10. else return select$(A_2, n-r, k-r)$ ——— $T(|A_2|)$

Deterministic selection

find median of $M \Rightarrow$ pivot

Suppose running time $T(n)$

$$\Rightarrow T(n) \leq \begin{cases} c_1 & n < 140 \\ T(\frac{n}{5}) + \max\{T(|A_1|) + T(|A_2|)\} + an + b & n \geq 140 \end{cases}$$

$T(|A_1|)$, $T(|A_2|)$ — How small can they be? $= 0$

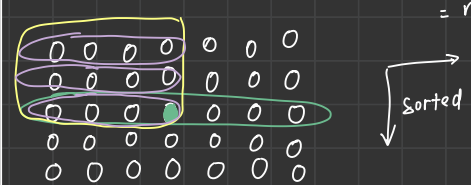
$\Rightarrow \max\{T(|A_1|), T(|A_2|)\} = \max(0, n) = n$

(ex)

Zone X

$<x$	45	70	Zone W
	60	80	
10 40	75	100	110 200
81	80	120	$>x$
160	150	250	

Zone Z — Zone Y



if $e \in$ zone $X \Rightarrow e < X \Rightarrow e \in A_1$

$|\text{zone } X| \leq |A_1| \Rightarrow |A_2| \leq n - 1 - |\text{zone } X|$

$|\text{zone } Y| \leq |A_2| \Rightarrow |A_1| \leq n - 1 - |\text{zone } Y|$

Deterministic selection sort:

$$T(n) \leq \begin{cases} c_1 & n < 140 \\ T(\frac{2n}{10}) + T(\frac{7n}{10}) + cn + b & n \geq 140 \end{cases}$$

☆ Use structural induction to prove: $T(n) = O(n)$, or $\exists c > 0$, $T(n) < cn$ for all $n \geq 1$

① Base case: for all $1 \leq n < 140$ sorting is used

It takes $O(140 \log_2 140) \leq d \times 140 \times 7.2 = 1008d$ for some $d > 0$.

d: constant associated w/ TC for a mergesort alg

So it suffices to choose $c \geq 1008d$ to satisfy $1008d \leq c \times n$

② Assumption: $T\left(\frac{2n}{10}\right) \le c\frac{2n}{10}$, $T\left(\frac{7n}{10}\right) \le c\frac{7n}{10}$

③ Induction: $n \ge 140$

$$
\begin{aligned}
T(n) &\le T\left(\frac{2n}{10}\right) + T\left(\frac{7n}{10}\right) + an + b \\
&\le c\frac{2n}{10} + c\frac{7n}{10} + an + b \\
&= c\frac{9n}{10} + an + b \\
&= cn - c\frac{n}{10} + an + b \\
&\le cn - c\frac{n}{10} + (a+b)n \\
&\le cn \quad \text{when we choose } c \ge 10(a+b)
\end{aligned}
$$

$\Rightarrow$ We have proven that for $c = \max\{10(a+b), 100bd\}$, $T(n) \le cn$ for all $n \ge 1$

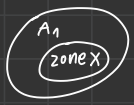
★ (3.2) Randomized algorithm analysis:

- with probability $\frac{1}{n}$, an element is picked.
- with a large chance (80%) that an element w/ rank between top 11 and 90 is picked.
- this element is used as a pivot for partition.
- after partition, with large chance, the sublist has length a fraction of n;

$\Rightarrow$ lead to average Time $\tilde{T}(n) = O(n)$ randomized selection alg.

A_1 = set of elements $e < X$ (pivot X)

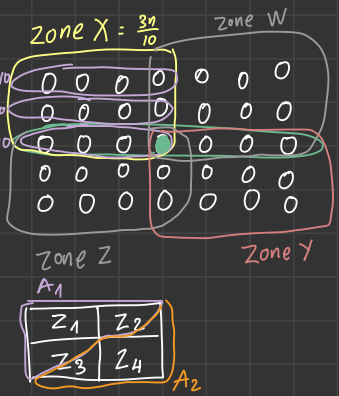
↳ rank(X) = median ●



zone $X = |X| = \frac{3n}{10} - 1$ (not counting pivot)

But in W and Z, we have not considered zone W and Z. So some elems in W, Z can be smaller than A_1

$\Rightarrow$ Length of X is smaller than A_1.



$|A_2| + |A_1| + 1 = n$ (pivot)

- $|X| = \frac{n}{10} \times 3 - 1 \Rightarrow |A_1| \geq |X| \Rightarrow |A_1| \geq \frac{3n}{10} - 1$

$\Rightarrow 1 - |A_2| - n \geq \frac{3n}{10} - 1$

$\boxed{\Rightarrow |A_2| = n - |A_1| - 1 \leq \frac{7n}{10}}$

- $|Y| = \frac{n}{10} \times 3 - 1 \Rightarrow |A_2| \geq |Y| \Rightarrow |A_2| \geq \frac{3n}{10} - 1$

$\boxed{\Rightarrow |A_1| = n - |A_2| - 1 \leq \frac{7n}{10}}$

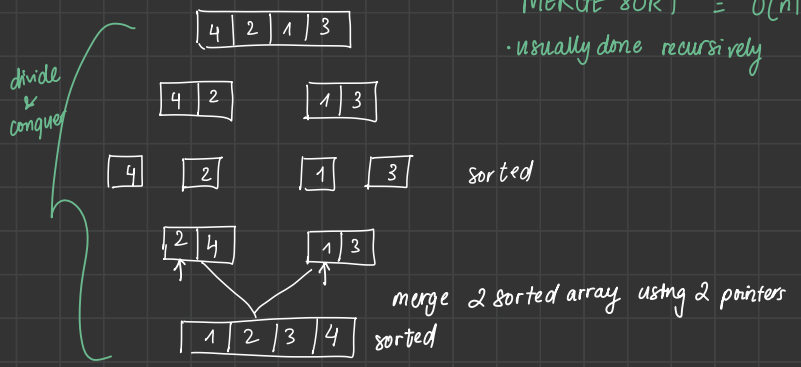
Since $\max\{|A_1|, |A_2|\} \leq \frac{7n}{10}$

$$T(n) \leq \begin{cases} c_1 & n < 140 \\ T\left(\frac{2n}{10}\right) + T\left(\frac{7n}{10}\right) + an + b & n \geq 140 \end{cases}$$

Youtube Review - Outside of lectures

MERGE SORT = O(nlogn)

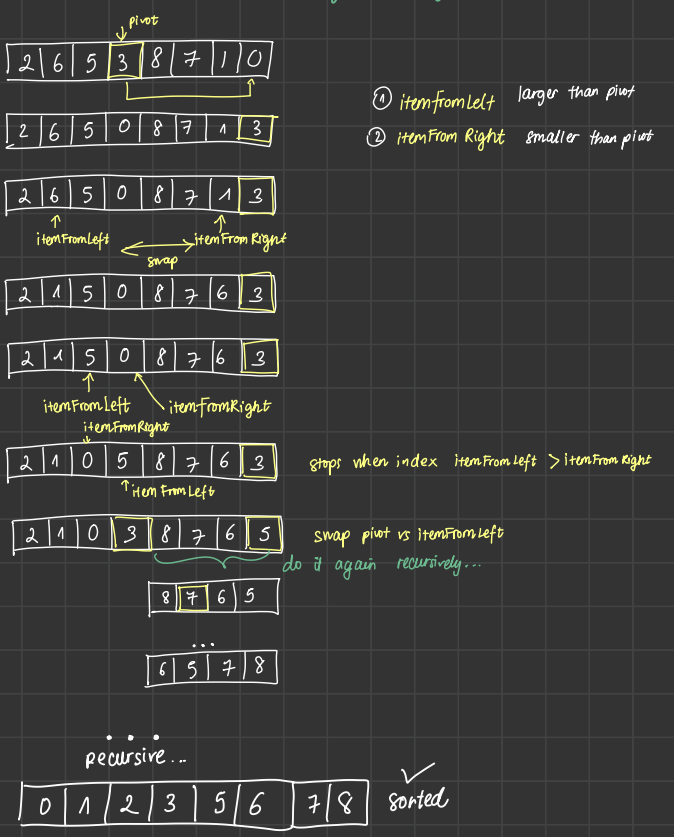
- usually done recursively



QUICKSORT

- recursive
- pivot

correct position in final array
items to left are smaller
items to the right are larger



☆ Deterministic Selection:

Partition $S = \{x_1, x_2, \dots, x_n\}$ in $\frac{n}{5}$ size groups

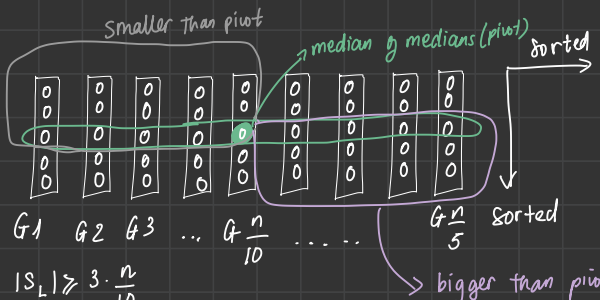
Subgroup $= SG_1 = \{x_1, x_2, x_3, x_4, x_5\}$ $\quad SG_2 = \{x_6 \dots, x_{10}\}$

$\dots$

$SG_{\frac{n}{5}} = \{x_{30}, x_{31}\} \rightarrow$ last subgroup

SG_1	5	15	8	3	(7)
SG_2	1	(9)	10	6	17
SG_3	12	(14)	2	16	10

$\Rightarrow S_{median} = \{7, 9, 14\}$ ← pivot; ↳ median of medians



So $|S_L| \geq 3 \cdot \frac{n}{10}$

$|S_R| \geq 3 \cdot \frac{n}{10}$

$|S_L| \leq n - |S_R| \leq \frac{7}{10} n \Rightarrow |S_R| \leq n - |S_L| \leq \frac{7}{10} n$

Deterministic algorithm:

$T(n) \leq T(\frac{7}{10}n) + T(\frac{n}{5}) + O(n)$ — $T(\frac{7}{10}n)$: worst case; $T(\frac{n}{5})$: finding median of medians

$\Rightarrow T(n) \leq T(\frac{7}{10}n) + T(\frac{2n}{10}) + O(n)$

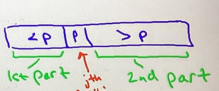
☆ Stanford Algorithms video:

Review on Randomized Alg:

Randomized Selection

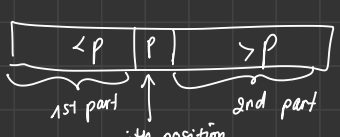
RSelect (array A, length n, order statistic i)

0. if n=1 return A[1]
1. choose pivot p from A uniformly at random
2. partition A around p; let j = new index of p
3. if j=i return p
4. if j>i return RSelect (1st part of A, j−1, i)
5. [if j<i] return RSelect (2nd part of A, n−j, i−j)



2 main cases:

- order of pivot > what we're looking for (i): RSelect (1st part of A, j−1, i)
- order of pivot < what we're looking for: RSelect (2nd part of A, n−j, (i−j)) ← looking for i−j
  ↳ discard anything less than p and p itself



⇒ Random pivots usually give pretty good splits. Average $\approx O(n)$

★ Deterministic Selection: → 50/50 split of partition → median

Recall: "best pivot" = median!

★ Choose pivots:

- logically break A into $\frac{n}{5}$ groups of size 5 each
- sort each group
- choose a median for each group
- copy $\frac{n}{5}$ medians (middle / 3rd largest) into new array C
- recursively compute median of C (!)
- return this as pivot

The DSelect Algorithm

DSelect(array A, length n, order statistic i)

1. Break A into groups of 5, sort each group — $O(n)$
2. C = the n/5 "middle elements" → $O(n)$
3. p = Select(C,n/5,n/10) [recursively computes median of C] — $T(\frac{n}{5})$
4. Partition A around p → $O(n)$
5. If j = i return p — c
6. If j < i return Select(1st part of A, j-1, i) — $T(?)$
7. [else if j > i] return Select(2nd part of A, n-j, i-j) — $T(?)$

choose pivot

same as randomized

recursive call TC depends on how good the pivot is

$$T(n) \leq cn + T\left(\frac{n}{5}\right) + T(?)$$

cn: sort + partition; $T(\frac{n}{5})$: recursive calls in line 3; $T(?)$: recursive call in 6 & 7

Solve T(?):

Key lemma: 2nd recursive call in line 6 and 7 guaranteed to be on an array of size $\leq \frac{7}{10}n$.

Goal: replace $T(?)$ with $T(\frac{7}{10}n)$

Rough proof: Let $k = \frac{n}{5}$ = # of groups

X_i = i^{th} smallest of the k middle elements

pivot = $X_{\frac{k}{2}}$

★ Partition around the pivot:

- key idea: partition array around a pivot elem.
- pick elem of array

3	8	2	5	1	4	7	6

- rearrange so that:
  - left of pivot ⇒ < pivot
  - right ⇒ > pivot

2	1	3	6	7	4	5	8

< pivot > pivot

★ If pivots are always chosen in the worst possible ways, the running time of randomized selection alg is: $O(n^2)$ (same as worst for quicksort)

suppose chose pivot = minimum for every time

★ Running time of Randomized Selection?

- depends on pivot
- average $\simeq O(n)$
- best case: the median! → $T(n) \leq T(\frac{n}{2}) + O(n)$
- "average" is over random pivot choices made by alg.

Quiz 3 Review - Old Exams

① True or False?

1. Quick sort is fast, it has worst running time $O(n \log_2 n)$

→ **False**. In some cases when the pivot happens to be the smallest elem (rare), the running time can be $O(n^2)$

2. Randomized quicksort runs in time $O(n \log_2 n)$ in average.

→ **True**

3. The only place uses randomness in randomized quicksort is when everytime randomized partition picks a pivot.

→ **True**

4. Quicksort can be solved in the worst case time $O(n \log_2 n)$ if it uses algorithm select to find the median and uses it as a pivot for partition

→ **True** When median is chosen as a pivot → partition arrays into 2 equal halves → $O(n \log_2 n)$

5. Running Randomized quicksort twice may consume 2 diff. amt of time.

→ **True**

ex) [12, 9, 7, 15, 10] (i, j, p)

⇒ [9, 7, 10, 15, 12]

All sorting algorithms

MERGESORT: $O(n \log_2 n)$

$$T(n) = \begin{cases} a & n = 1 \\ 2T\left(\frac{n}{2}\right) + bn & n \geq 2 \end{cases}$$

QUICKSORT: deterministic + randomized

- $O(n \log_2 n)$ ideal case – average
- $O(n^2)$ worst case

- Worst case: $T(n) = T(n-1) + bn$ → when bad pivot (smallest elem)
- Average (ideal) case: $T(n) = 2T\left(\frac{n}{2}\right) + bn$ → when pivot is median
- General: $T(n) = (n-1) + \frac{2}{n} \sum_{i=0}^{n-1} T(i)$

SELECTION SORT:

☆ Deterministic Selection: $O(n)$ worst case

$$T(n) \leq \begin{cases} c_1 & n < 140 \\ T\left(\underbrace{\frac{2}{10}n}_{\text{from } \frac{n}{5}}\right) + T\left(\frac{7n}{10}\right) + an + b & n \geq 140 \end{cases}$$

☆ Randomized Selection: – Average : $O(n)$

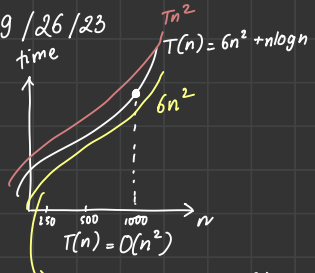
– worst case : $O(n^2)$ – when pick the largest/smallest elem on each iteration.

Quiz Review huhu:

a) 30 50 20 40 60 80 70

④ Complexity lower bounds

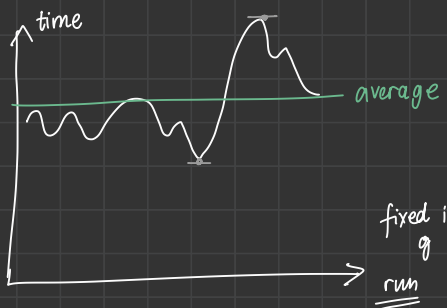
09/26/23



lower bound $\Rightarrow T(n) = O(n^2)$

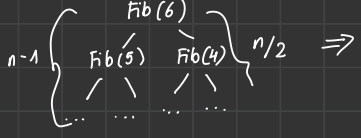
We always look for worse case time. For low complexity we don't really care.

I have 2 funcs upper bound & lower bound (omega) that we can sandwitch the TC in.



LOWER BOUND
#
BEST CASE!!

ex: Rec-Fibonacci(N)



$\Rightarrow$

How big is this tree's size?

$T(n) \leq T(n) \leq T(n)$

$\approx 2^n - 1 \Rightarrow$ total # nodes

$\Rightarrow 2^{\frac{n}{2}} \lesssim T(n) \lesssim \approx 2^n - 1$

$\Rightarrow T(n) = O(2^n)$

$\Rightarrow T(n) = \Omega(2^{\frac{n}{2}})$

- Big O: - sufficient time - upper bound
- Big Ω - necessary - lower bound

4. Complexity lower bounds

- Complexity upper bound: $T(n) \leq U(n) \Rightarrow T(n) = O(U(n))$
- Complexity lower bound: $L(n) \leq T_{wc}(n) \Rightarrow T(n) = \Omega(L(n))$

(ex) Insertion sort:

- upper bound: $T(n) = O(n^2)$, can we say $T(n) = O(n^3)$?

$T(n) = O(n^2)$

$\Rightarrow \exists c, n_0$ such that

$T(n) \leq cn^2$ where $n \geq n_0$

$\Rightarrow cn^2 \cdot n$ when $n \geq \max\{n_0, 1\}$

$\Rightarrow \boxed{T(n) = O(n^3)}$ → yes

Definition 1:

- Let $f(n), g(n)$ be 2 functions in n

$f(n) = \Omega(g(n))$ if there are constants $c > 0$, $n_0 > 0$ s.t.

$f(n) \geq cg(n)$ when $n \geq n_0$

ex: $f(n) = 2n + 3$

$2n + 3 \geq 1 \cdot n$, $n \geq 1$

$f(n) = \Omega(n)$ ✓

or also true $2n + 3 \geq 1 \cdot \log(n)$

$\Rightarrow f(n) = \Omega(\log n)$

Definition 2: $f(n) = \Theta(g(n))$

if $f(n) = O(g(n))$ & $f(n) = \Omega g(n))$

e.g. if $f(n) = O(n^2)$ $f(n) = \Omega(n^2)$

then $f(n) = \Theta(n^2)$

$f(n)$ $g(n) = n^2$

$2n^2 \leq 2n^2 + 3n \leq 5n^2$

$f(n) = 2n^2 + 3n$

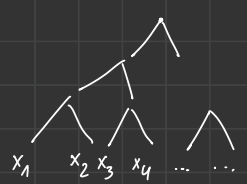
$= \Theta(n^2)$

$$T(n) = \begin{cases} a & n = 1 \\ T(\frac{n}{2}) + b & n \geq 2 \end{cases}$$

Prove: $T(n) = \Omega(\log_2 n)$ using induction

"$\exists c, n_0$ st $T(n) \geq c \log_2 n$"

(ex) Find $\max(x_1, x_2, \ldots, x_n)$



$n-1$ sufficient

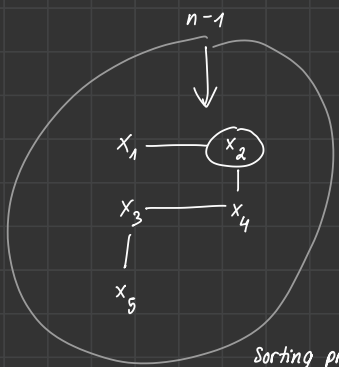
$n-1$

x_1 — x_2

x_3 — x_4

$\frac{n}{2}$ x_5 — x_6

⋮ ⋮



Another example:

"time" — # of basic operations $\times$ + −/

comparisions between elements

if $i = 1$ $x_i \leq x_j$

abstract models → "comparision-based"

Sorting problems have lower bound $\Omega(n \log_2 n)$ on comparision-based models.

① Lower bound of an algorithm:

Lower bound $L(n)$ for time func $T(n)$ is s.t:

$$T(n) = \Omega(L(n))$$

For example, we can prove for Binary Search $T(n) = \Omega(\log_2 n)$ by having $c>0, n_0>0$ s.t. $T(n) \geq c\log_2 n$ holds for $n \geq n_0$

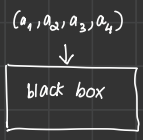
② Lower bound of a problem

- Lower bounds can be established for certain problems, regardless of algorithms

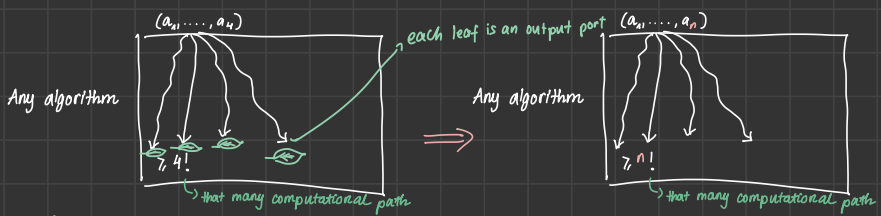
09/28/23

Claim: Sorting problem has lower bound $\Omega(n \log_2 n)$ on comparision-based models (comparisons are done btw elements)

- We exam "any given algorithm"
- It is treated as a blackbox, with input L and output sorted L

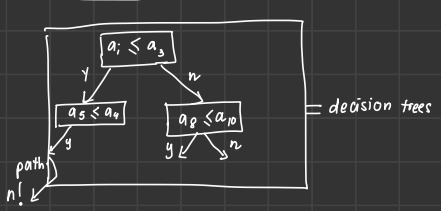


There are 4! scenarios of inputs



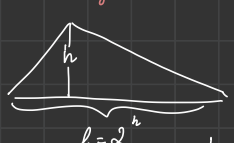
Comparison-based:

$a_i \leq a_j$



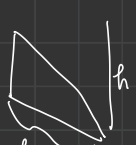
decision trees

Number of leaves = $n!$



$h = \log_2 l$

$= \log_2 2^h = 2$



⇒ The longest path length in the shortest-depth binary tree of l leaves

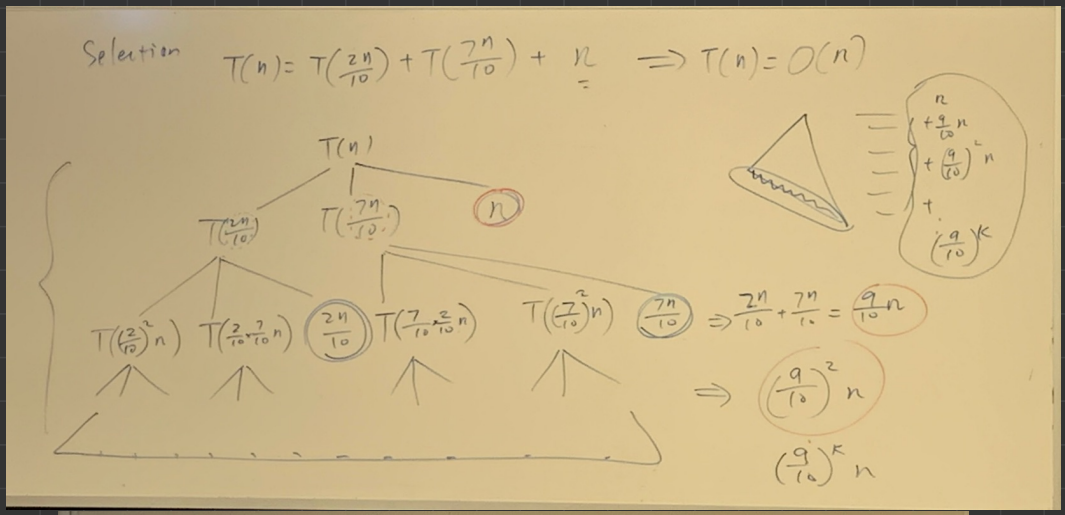
worst case time $\geq \log_2 l$

$$n! = \underbrace{1 \times 2 \times 3 \times \dots \times \left(\frac{n}{2}\right)}_{\geq \left(\frac{n}{2}\right)^{\frac{n}{2}}} \times \left(\frac{n}{2}+1\right) \times \dots (n-1) \times n$$

$$\geq \left(\frac{n}{2}+1\right) \times \underbrace{\left(\frac{n}{2}+2\right)}_{\geq \frac{n}{2}} \times \dots \times \underbrace{(n-1)}_{\geq \frac{n}{2}} \times \underbrace{n}_{\geq \frac{n}{2}}$$

$$\geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

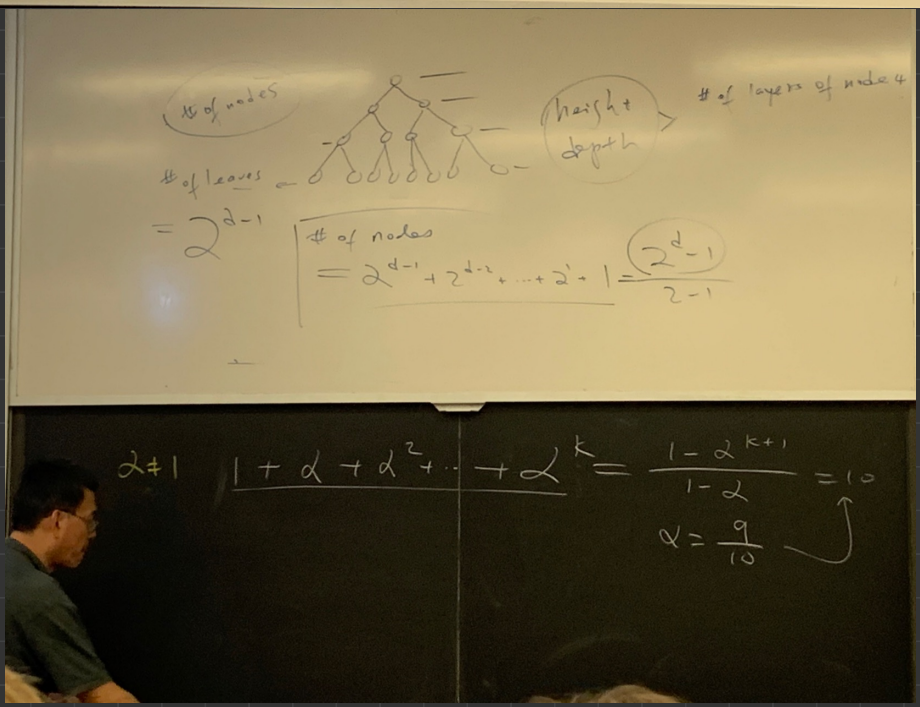
Selection $T(n) = T\left(\frac{2n}{10}\right) + T\left(\frac{7n}{10}\right) + n \Rightarrow T(n) = O(n)$

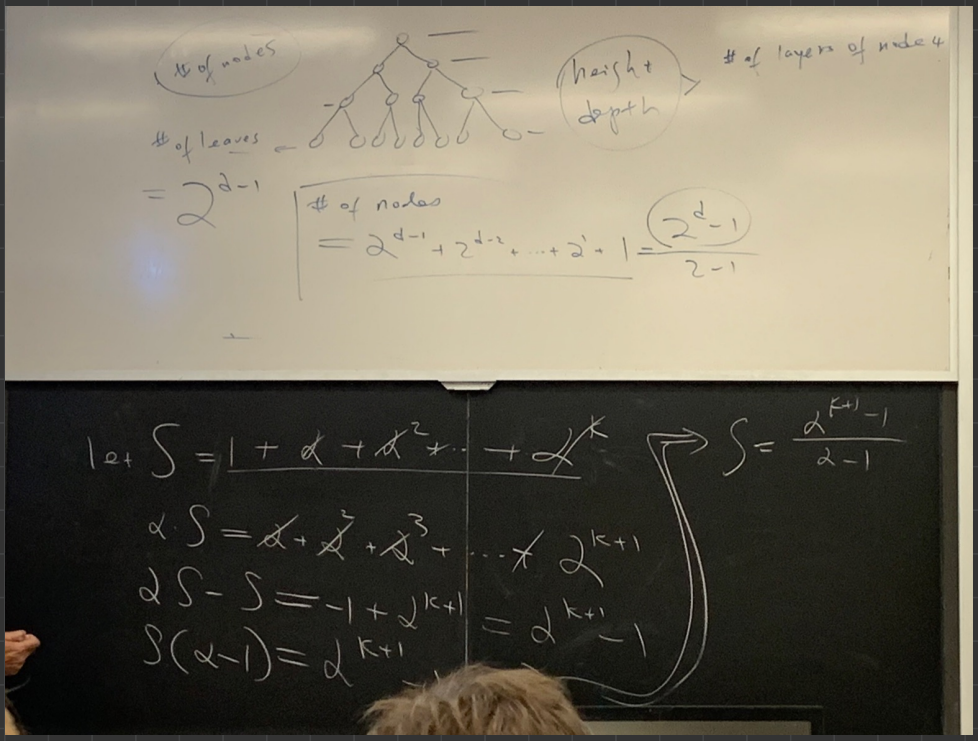


$\Rightarrow \frac{2n}{10} + \frac{7n}{10} = \frac{9}{10}n$

$\Rightarrow \left(\frac{9}{10}\right)^2 n$

$\left(\frac{9}{10}\right)^k n$





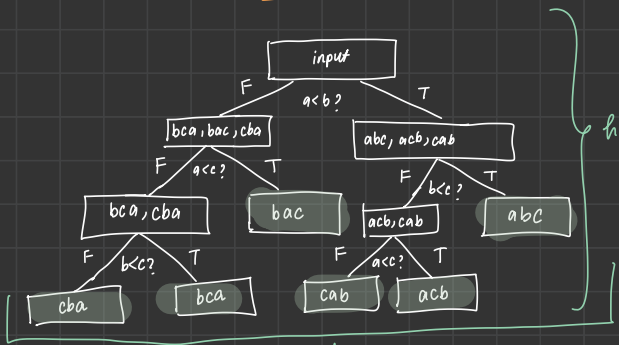
2 Quiz on Monday :)

Youtube Sorting Lower Bound (Problem-based) not alg-based!!!

- answers the question if it's better to do better...

⇒ Sorting: many are comparision-based

Decision tree example on insertion sort, $n = 3$: a, b, c



Proof:

There are $n!$ leaves. A tree of height h has at most 2^h leaves.

$2^h \geq n!$

$h \geq \log_2(n!)$

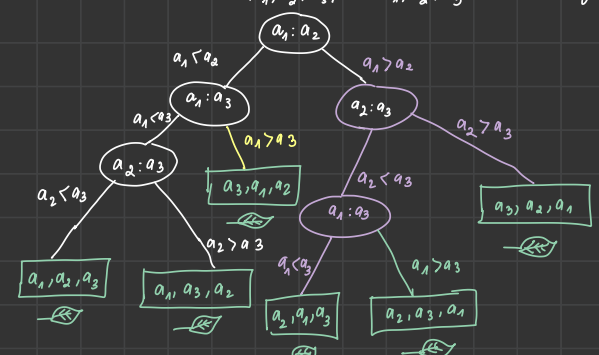
$\geq c \cdot \log_2(n^n)$ } why? Next page is proof

$\geq c \cdot n \log_2(n)$

$\Rightarrow h \in \Omega(n \log n)$

Youtube Lower bound on comparision based sorting $\Omega(n \log n)$

Let n have 3 numbers $\langle a_1, a_2, a_3 \rangle \rightarrow a_1, a_2, a_3$ can be arranged in $3! = 6$ ways



n numbers = n! permutation

DECISION TREE

In this tree, all permutations are formed at leaves of trees

For n elems, n! perms.

⇒ A tree w/ height h has as max 2^h leaves

Let ℓ = number of leaves in tree ⇒ $n! \leq \ell \leq 2^h$

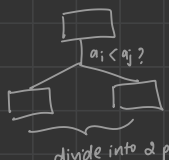
$\Rightarrow h \geq \log(n!)$

$\Rightarrow h = \Omega(n \log n)$

⇒ If we have comparision based sorting alg on input size n, there's at least some input that makes it do at least nlogn work

the # of comparisions needed to get to bottom of tree

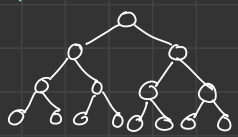
Proof:



# of leaves
- min: $n!$
- max: 2^h

h: number of comparision
2^h: # leaves in binary tree

Prove: $2^h \geq n!$



!!! Important

$n!$ ≤ 2^h → thinking of this as # leaves in decision trees cannot be > than # leaves in generic binary tree

$n!$: decision tree leaf nodes
2^h: max leaf nodes in binary tree

$\Rightarrow 2^h \geq n!$

$\Rightarrow \log_2(2^h) \geq \log_2(n!)$

$\Rightarrow h \geq \log_2(n!)$

$\Rightarrow h \geq \log_2(\frac{n}{2}^{n/2})$ ← get rid of $\frac{1}{2}$ in $\frac{n}{2}$

$\Rightarrow h \geq \frac{n}{2}\log_2(\frac{n}{2})$

$\Rightarrow h \geq \frac{1}{2} n \log_2(\frac{n}{2})$

$\Rightarrow h \geq c \cdot n \log_2(n)$

$\Rightarrow \boxed{h \geq c \cdot n \log_2(n)}$

Thus, $h \in \Omega(n \log n)$

$n! \in \Omega(n^n)$ ← take first half

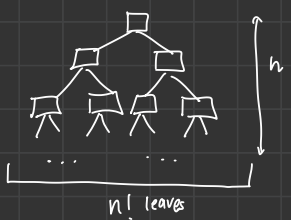
$n! = \underbrace{n \cdot (n-1)(n-2)}_{\geq n \cdot (n-1)(n-2)} \cdots 2 \cdot 1$

$\geq n \cdot (n-1)(n-2) \cdots (\frac{n}{2})$

$\geq \frac{n}{2} \cdot \frac{n}{2} \cdot \frac{n}{2} \cdots \frac{n}{2}$ → take the smallest value: $\frac{n}{2}$

$= (\frac{n}{2})^{n/2}$

$\Rightarrow \boxed{n! \geq (\frac{n}{2})^{n/2}}$



Quiz 3 Review:

① function Work(L, i, j)

```
if (i<j)        ___ c
    swap(L[i], L[j])   ___ c
    Work(L, i+1, j-1)  ___ T(n-2)
    return
```

[8, 1, 2, 3, 6, 4]
⇒ [4, 1, 2, 3, 6, 8]
[4, 6, 2, 3, 1, 8]
[4, 6, 3, 2, 1, 8]

① what does algorithm Work do?

Reverse the list (w/ 2 pointers)

② T(n) is worst time complexity for Work on input size n. Express n iterms of i & j ⇒ $n = j - i + 1$

③ Recursive form of T(n):

$$T(n) = \begin{cases} c & \text{base case : when } n = 1 \\ T(n-2) + c & \text{recursive cases when } n > 1 \end{cases}$$

② TC of QuickSort based on various scenerios of the selected pivot. Give recursive formulation for T(n), for n > 1

① On the worst pivot, $T(n) = T(n-1) + bn$ → when bad pivot like smallest elem

② On ideal case pivot, $T(n) = 2T\left(\frac{n}{2}\right) + bn$ → when median is pivot

③ On averaged case pivot, $T(n) = (n-1) + \frac{2}{n}\sum_{i=0}^{n-1} T(i)$

* if elem at j < pivot
i = i + 1
exchang i & j

③ [30, 70, 50, 20, 80, 40, 60] is input of randomized quick sort

① If pivot = [60], the partition function on this list should return? ⇒

[30, 70, 50, 20, 80, 40, (60)] ← pivot

after partition → [30, 50, 20, 40, 60, 70, 80]

b) [30, 70, (50), 20, 80, 40, 60]
⇒ [30, 20, 40, 80, 60, 70]

④ Selection sort: on input (A, n, k), the algorithm finds k^{th} smallest elem in set A containing n elements. Specially, it first finds a pivot x and then uses x to partition set A into two subsets A_1 and A_2, so that the k^{th} smallest elem is recursively found from either of the subsets.

① To find pivot x, the algorithm recursively calls Selection$\left(M, \frac{n}{5}, \frac{n}{10}\right)$ where M is the array containing all medians of groups and x is the median of M.

② If rank(x) = r > k, recursive call Selection$(A_1, r-1, k)$ is executed.

if r < k, call Selection$(A_2, n-r, k-r)$ is executed.

③ Selection algorithm has TC:

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + bn$$

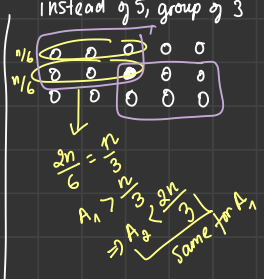
where $\frac{n}{5}$ is size of M and $\frac{7n}{10}$ is upper bound for A_1 and A_2.

$\max\{|A_1|, |A_2|\}$ (A_1: length of first half, A_2: length of second half)

- If algorithm uses groups of 3 elements instead of 5, T(n) would be:

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + dn, \text{ for some } d > 0.$$

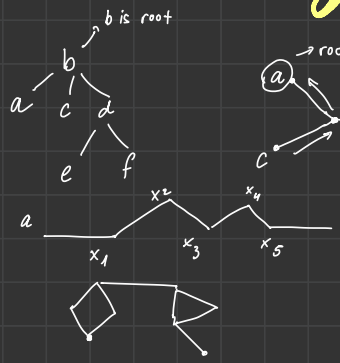
instead of 5, group of 3



algorithms on graphs

★ Topics:

- Basic + representation of graphs
- Depth-first search & applications
- Shortest path algorithms
- priority queue



Bi-partite graphs

$G = (V, E)$

$G = (V_1 \cup V_2, E)$

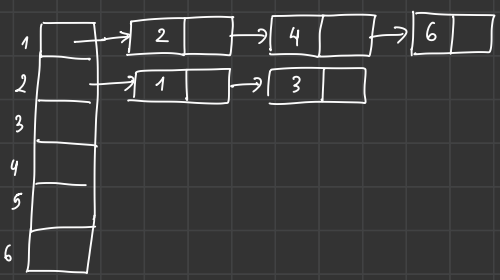
$E \subseteq V_1 \times V_2$

Complete graphs

10/2/23

★ Fundamentals of graphs

$G = (V, E)$ $V = \{1,2,3,\dots,6\}$ $E = \{(1,2),(2,3),(1,6),\dots\}$



Dense $|E| = \Omega(|V|^2)$

Sparse $|E| = O(|V|^{1+\epsilon})$

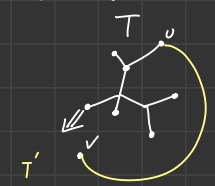
Trees:

(1) a single vertex is a tree

(2) let T be a tree

v is not in T

u is in T



Then adding v and (u,v) to T yeids a tree.

(V, E)

$\{x\}$ $\emptyset$

$G = \{$ all non-empty tree$\}$

★ Recursively define a set of trees & graphs

- set pair $(\{x\}, \emptyset)$ is in G.
- if (V,E) is in G, $U \subseteq V$, and $x \notin V$,

then (V', E') is in G, where

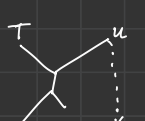
$V' = V \cup \{x\}$

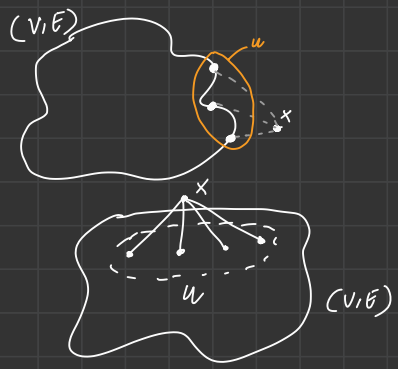
G = set of all non-empty trees

(1) $(\{v\}, \emptyset)$ is in T

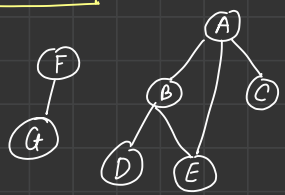
(2) if (V,E) is in T, $v \notin V$, $u \in V$

then $(V \cup \{v\}, E \cup \{(v,u)\})$ is in T



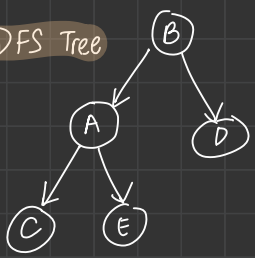


Depth first search:



all reachable nodes are gonna be visited

DFS Tree



function DFS (G)

1. ∀x , visited (X) ⇐ false
2. ∀x , if visited(x) ⇐ false
   call explore (G,x)

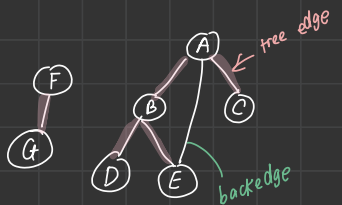
DFS Pseudo Code

Assume graph G with x being any vertex → Explore all vertices reachable from vertex x.

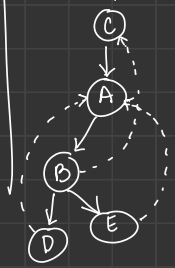
```
func explore(G:graph, x:vertice)
1. visited (x) = true
2. for each edge (x,y) in G
3.     if not visited (y), explore (G,y)
```

Adding time stamp :

```
func explore (G: graph, x: vertex)
1. visited (x) = true
2. pre (x) = time_stamp          → pre-visit work
3. time_stamp = time_stamp + 1
4. for each edge (x,y) in G
5.     if not visited (y)
6.         parent (y) = x        → record tree edge
7.         explore (G,y)
8. post (x) = time_stamp         → post visit work
9. time_stamp = time_stamp + 1;
```

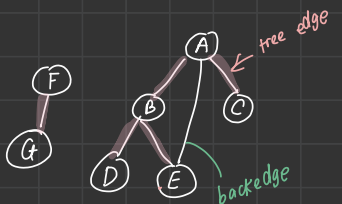


DFS Tree forest

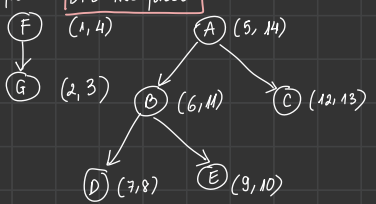


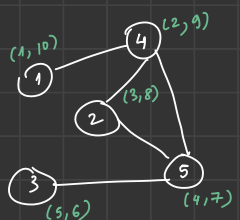
tree edge = parent → child

backedge : descendent → ancestor



★ Starting from F: DFS tree forest





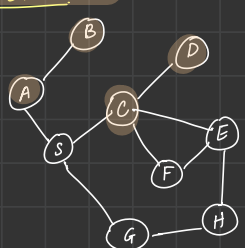
Output DFS:

1 4 2 5 3 (visited)

3 5 2 4 1 (complete)

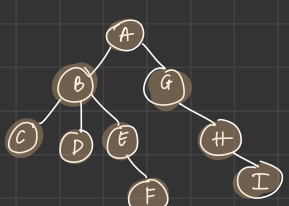
Youtube learn:

ex:

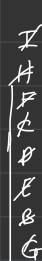


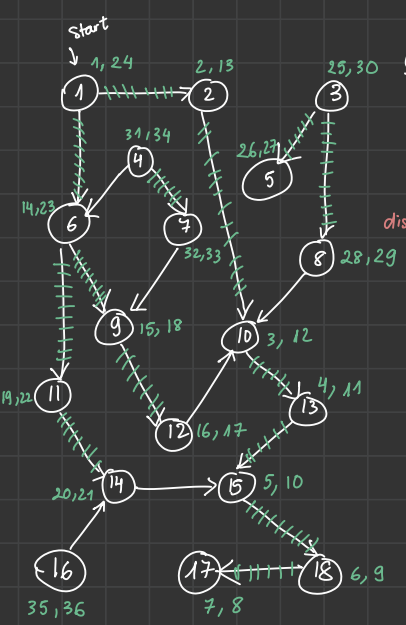
A B S C D E H G F

DFS: depth = vertical before horizontal



Stack - to be visited





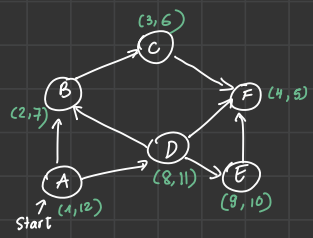
Colors:

(empty): white

#, _ : gray

#, # : black

(discovered, finished)

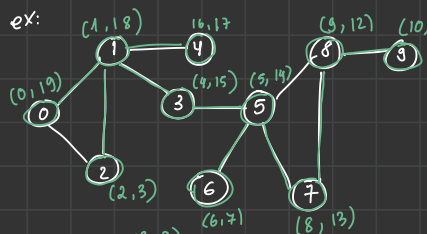


all nodes have order pairs

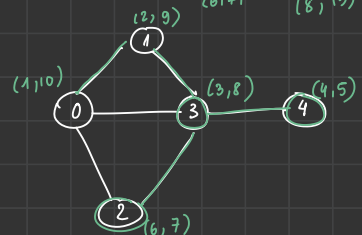
DFS Pre-order: A B C F D E

DFS Post-order: F C B E D A

ex:

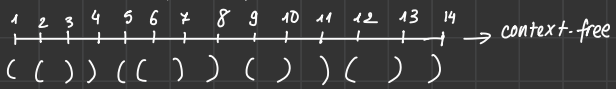


⇒ DFS order: 0 1 2 3 5 6 7 8 9 4



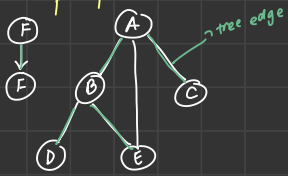
DFS(G, 0): 0 1 3 4 2

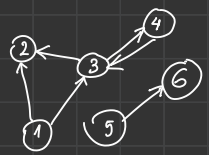
nested parallel pathesis



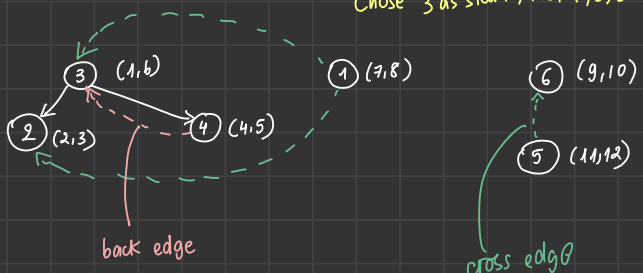
DFS: 1 2 4 6 3 5 7 8

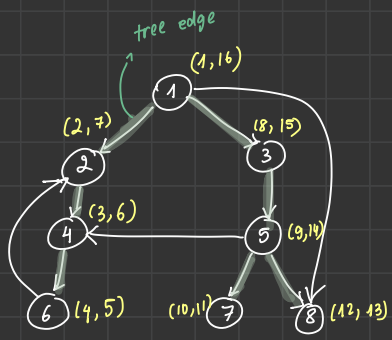
★ Example for DFS:



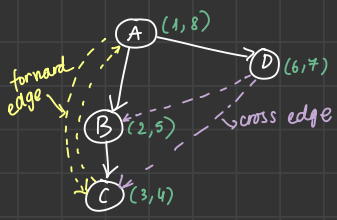


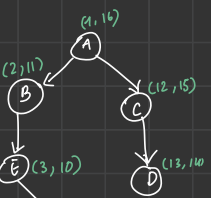
Chose 3 as start, then 1, 6, 5



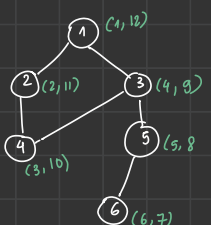


Backedge: 6 → 2
Forward edge: 1 → 8
Cross edge: 5 → 4



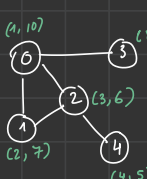


ex:



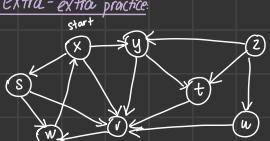
Notes

pre(x) < pre(y) < post(y) < post(x)

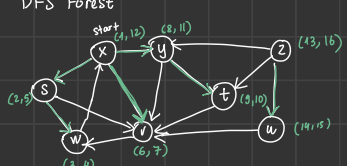


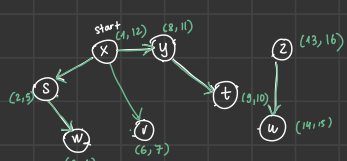
0 1 2 4 3

Extra-extra practice



DFS Forest



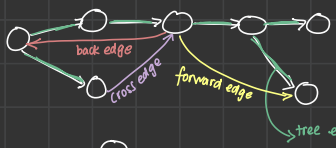


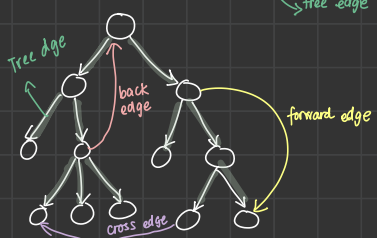
★ Types of edges in DFS tree:

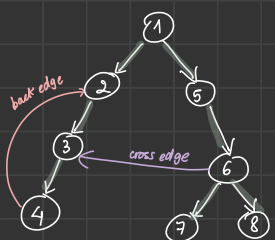
- Tree edge : edge which is present in the tree obtained after applying DFS on the graph.
- Back edge: edge (u,v) s.t. v is the ancestor of node u but is not part of DFS tree
- Forward edge: edge (u,v) such that v is a descendent but not a part of DFS tree.
- Cross edge: edge that connects two nodes s.t. they do not have any ancestor and descendant relationship between them.

ex: ★During DFS, classification of edge (u,v):

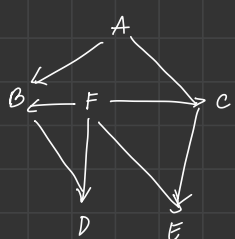
- If v is visited for the first time as we traverse the edge (u,v), then (u,v) is a **tree edge**
- Else, v has already been visited:
  - a) v is ancestor of u, then (u,v) is **back edge**
  - b) v is descendant of u, (u,v) is a **forward edge**
  - c) if not a) or b), then (u,v) is **cross edge**



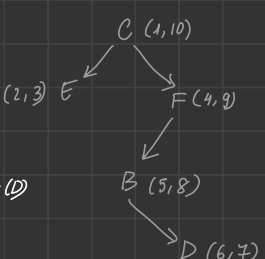




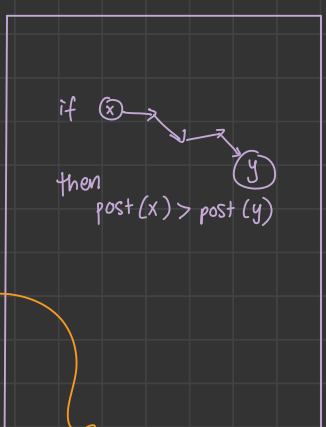
10/05/23



A ≻ D
post(A) > post(D)



A (11,12)



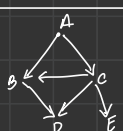
Application of DFS:

1. determine if the input graph is connected
2. determine if 2 given vertices are connected on input graph
3. determine if input graph contains a cycle.
4. find topological order of vertices on input DAG.
5. find the strong connected components of input graph
6. find single-source shortest paths on the input DAG

⇒ All algorithms have TC: $O(|E|+|V|)$

```
func DFS-main(G)
  1. For x in G, visited(x) = false,
  2. For x in G,
       if visited(x) = false
         explore(G,x)
```

A → B
≡ ¬B → ¬A

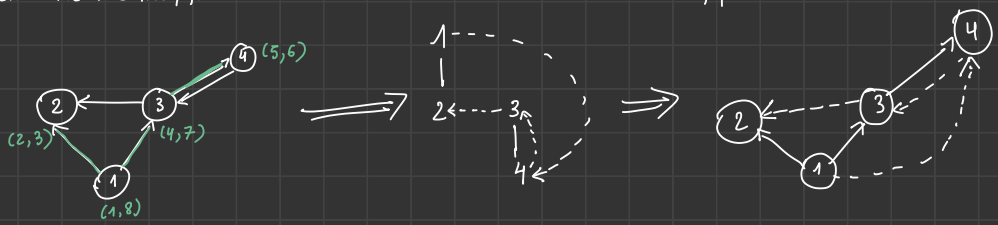


★ Topological sort order: (not DFS)

A C E B D
A C B E D
A C B D E

Graph Review For Quiz:

1) Show the DFS tree / forest:



2) Tree edges: (1,2), (1,3), (3,4)

Back edge: (4,3)

Forward edge: (1,4)

Cross edge: (3,2) ?

3) Time stamp:

(1): (1,8) (2): (2,3) (3): (4,7) (4): (5,6)

Midterm on Tuesday:

- Big O
- Recursive formulation of T(n)
- Induction
- Divide & conquer, Quick Sort, Selection
- Average case time
- Lower bound
- Lower bound proof for sorting
- DFS forest
- Topological sort + SCC

10/09/23

Quiz 4 Solution

① These questions concern TC lower bound for algorithms and lower bound for problems. Choose T or F.

a) A lower bound of an algorithm is the amount of time required (↳ key word) by the algorithms to solve worst case input instances.

⇒ True

b) Some algorithms have lower bound $\Omega(n^2)$ and upperbound $O(n\log_2 n)$

⇒ False lower bound in algorithms cannot be higher than upper bound.

c) For merge sort algorithm, because the worst case instances can be solved in $O(n\log_3 n)$ time, it allows us to conclude that the algorithm has lower bound $\Omega(n\log_3 n)$

⇒ False Cannot conclude, could be n.

d) If an algorithm solving some problem P has lower bound $\Omega(p(n))$, then the problem P has lower bound $\Omega(p(n))$

⇒ False

e) If an algorithm solving some problem P has upperbound $O(q(n))$, then the problem P has upper bound $O(q(n))$

⇒ True

② Comparision-based sorting lower bound: $\Omega(n\log n)$

a) In this comparision-based model, every comparison is done between 2 elements in the input only.

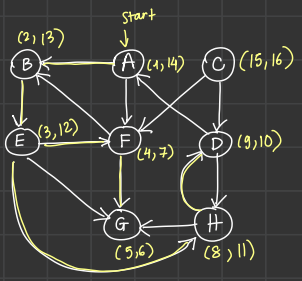
b) The number of diff. scenarios for an n-element list to any alg is $n!$

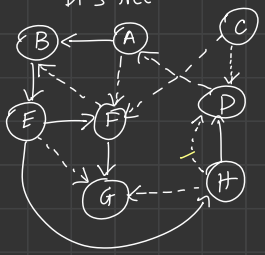
c) b) implies that there are $n!$ number of leaves in the binary decision tree that models the algorithm.

d) $x \geq \log_2 y$ (x: height of tree; y: number of leaves in tree)

e) Quantity $\log_2(n!)$ is the number of comparisons on a longest, shortest path of a deepest, shallowest decision tree model.

(3)





(4)

a) Every subsequent iteration after the first iteration, there exists 1 vertex that has not been visited yet

Midterm Review

① $T(n) = \begin{cases} 1 & \text{when } n = 1 \\ T(n-1) + n & n \geq 2 \end{cases}$

To prove $T(n) = O(n^2)$, we try to prove $\exists c$ and n_0 such that $T(n) \leq cn^2$ for $n \geq n_0$

* Base case: $n = 1 \Rightarrow T(n) = 1$, can be satisfied with choice $c \geq 1$

$= 1 \cdot c$

* Assumption: $T(n-1) \leq c(n-1)^2$

* Induction:

$T(n) = T(n-1) + n$

$\Rightarrow T(n) \leq c(n-1)^2 + n$

$\leq c(n^2 - 2n + 1) + n = cn^2 - 2cn + c + n$

$\leq cn^2 - 2cn + c + \underline{cn} \quad (c \geq 1)$

$= cn^2 \underbrace{-cn + c}_{\text{must be } < 0 \text{ since } n \geq 1}$

$\leq cn^2$

$\Rightarrow$ Thus, proven when $c \geq 1$ and $n_0 = 1$ $(n \geq 1)$

② Sorting problem is proved to have $\Omega(n \log_2 n)$ w/ decision tree based model

1) The lower bound $\Omega(n \log n)$ is w/ respect to comparision based model only.

2) n input → n! leaves in tree

3) For any decision tree, if # of leaves = ℓ, the height is $\geq \log_2 \ell$

4) Height of decision tree = worst case time of the algorithm that is modeled by the decision tree.

5) Lower bound $\Omega(n \log n)$ means there is no better alg than mergesort for the sorting problem in terms of number of element-element comparisions.

③ True or False?

Repeated Stuff! Skip if needed

1) quick sort is fast. worst TC = $O(n \log_2 n)$ → False. when any is sorted [ ] very bad uneven split

$\Rightarrow$ worst $O(n^2)$

2) Randomized quicksort runs $O(n \log_2 n)$ in average. → True

3) Quick sort can be solved in worst case $O(n \log_2 n)$ if use select to find median and uses it as pivot for partion

→ True

4) Running Randomized quicksort twice may consume 2 diff amt of time

Topological sort order:

Let $G=(V,E)$ be a DAG $\quad V=\{v_1, v_2, \dots, v_n\}$

a topological sort of V is a rearrangement of vertices in V $v_1', v_2', \dots, v_n'$ such that for every $i<j$, there is no directed path $v_j \to v_i$

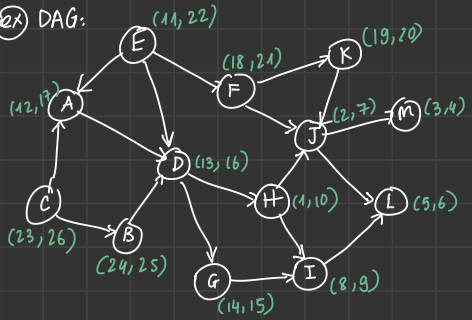
Topological sort: Input: direct acyclic graph DAG $G(V,E)$

Output: vertices of V in order: $v_1, v_2, v_3, \dots, v_n$ s.t. $\forall i,j$, $(v_j, v_i) \notin E$.

How? - Pick an unvisited node

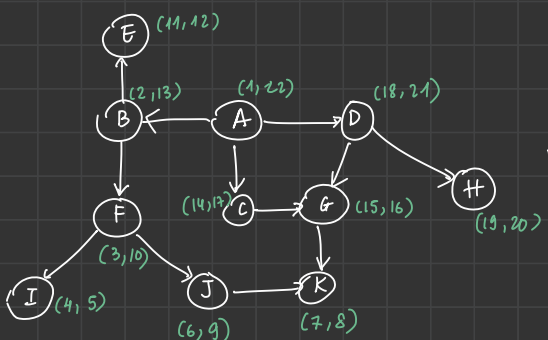
- From that node, do a DFS exploring only unvisited nodes
- On the recursive callbacks of DFS, add the current node to the topological ordering in reverse order

(2x) DAG:



Pick H as start, then E, then C

Topological ordering: C B E F K A D G H I J L M



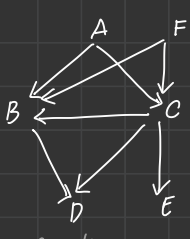
Topological Sort:

A D H C G B E F J K I

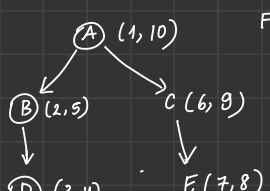
ex:

F A C E B D

(in videos)



graph



search tree

F (11,12)

tree edge: parent → child

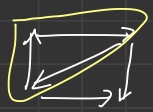
back edge: descendent → ancestor

cross edge: between siblings

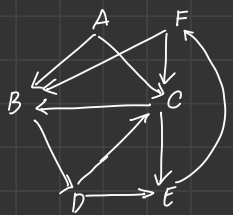
★ Strongly connected components

Let $G = (V, E)$ be a directed graph

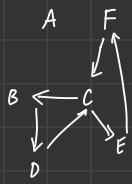
- Strongly connected components (SCC) is a maximal subgraph $H = (U, F)$, $U \subseteq V$, $F \subseteq E \cap U \times U$ such that $\forall x, y \in U$, there is a direct path $x \rightsquigarrow y$ and there is a directed path $y \rightsquigarrow x$.

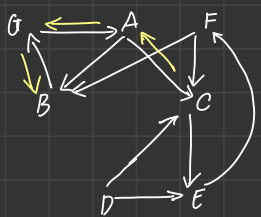


Then: every directed graph can be partitioned into SCC.

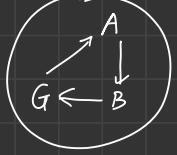


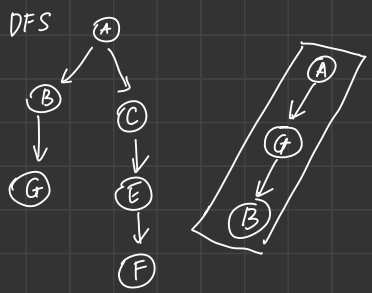
graph



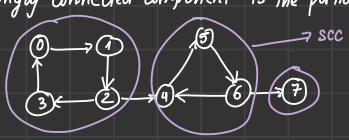


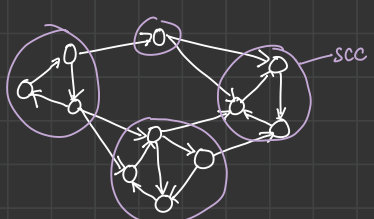
graph





★ A strongly connected component is the portion of a directed graph in which there is a path from each vertex to another vertex.





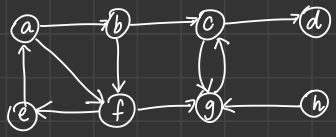
★ Finding SCC of directed graphs using Kosaraju's algorithm

1. Compute transposed graph G^r (all directions reversed)
2. run DFS on G^r → obtain postorder (finishing time for each vertex)
3. run DFS on G from vertex ✓ with highest post(v) value
   - processing nodes in decreasing order of finishing times

⟹ Get SCCs = nodes that share common leader

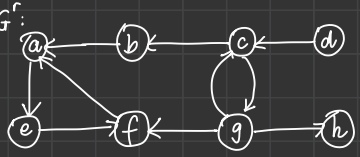
TWO PASSES OF DFS

ex G:



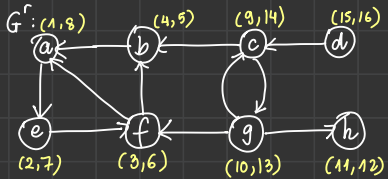
reverse G →

G^r:



→ transposed G

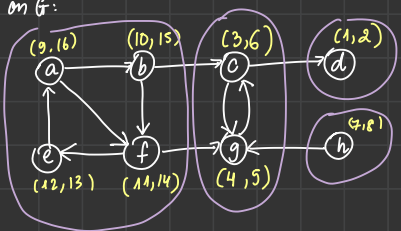
do DFS on G^r:



→ list of descending finishing time vertex:

d c g h a e f b → guide for DFS on original G.

DFS on G:



⇒ 4 SCCs in the graph: {a,b,e,f}, {c,g}, {d}, {h}

★ BFS!!!

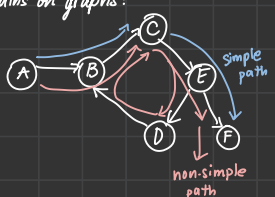
3. Shortest path problems

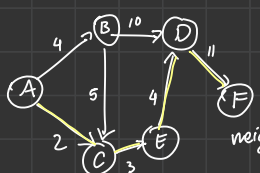
DAG: directed + acyclic

Dijkstra's: directed, non-negative edges + cycles

Bellman Ford: can contain negative edges

- Paths on graphs:

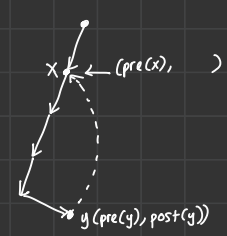


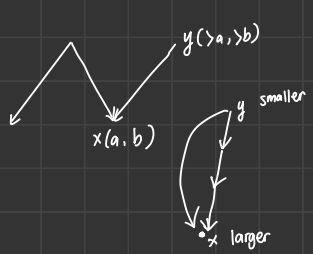


weight: 2+3+4+11 = 20

★ Problems about paths:

- Reachability: given $G=(V,E)$ and vertices $s,t \in V$, asked if there is a path $s \rightsquigarrow t$, from s to t.
- Cycle: given $G=(V,E)$ asked if there is a cycle in the graph.
- s-t shortest path: given $G=(V,E)$, vertices $s,t \in V$, find the shortest path $s \rightsquigarrow t$.
  - ↳ single-source shortest path: given $G=(V,E)$, $\forall v \in V$, find a shortest path $s \rightsquigarrow v$.
  - ↳ all pair shortest path: given $G=(V,E)$, $\forall u,v \in V$, find shortest path $u \rightsquigarrow v$



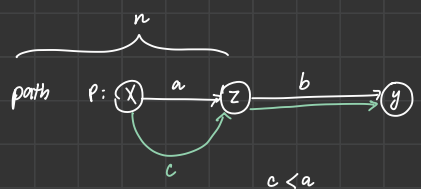


i) pre(y) > pre(x)

ii) post(x) no value yet

i) possibly exponential # of paths from s to t

ii) any subpath of a shortest path is a shortest path



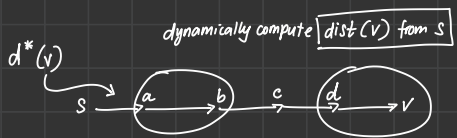
$c < a$

$c + b < a + b$

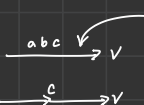
✱ Shortest distance problem:

- Input: di-graph $G = (V, E)$, weights $w: E \to R$; source $s \in V$
- Output: dist(v), shortest distance s to every vertex $v \in V$.
  - G is a DAG : DFS-based algorithm.
  - G doesn't contain negative edges : Dijkstra's algorithm.
  - G may contain negative cycles: Bellman-Ford algorithm.

✱ Shortest distance problem on DAG:



edge (d, v) is relaxed

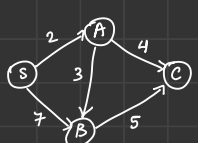


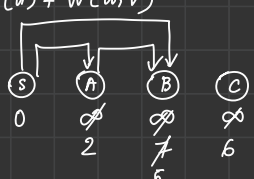
(d, v) has not been considered constraint

✱ shortest distance problem on DAG:

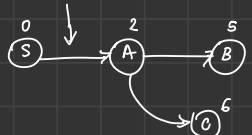
- updating path distances, called relaxation

$$dist(v) = \min \begin{cases} dist(v) \\ dist(u) + w(u, v) \end{cases}$$





s is predecessor of A



$dist(s) = 0$

$$dist(A) = \min \begin{cases} dist(A) & \infty \\ \underbrace{dist(s)}_{0} + \underbrace{w(s, A)}_{2} & = 2 \end{cases}$$

$$dist(B) = \min \begin{cases} dist(B) & \infty \\ \underbrace{dist(s)}_{0} + \underbrace{w(s, B)}_{7} & = 7 \end{cases}$$

$$dist(B) = \min \begin{cases} dist(B) & 7 \\ \underbrace{dist(A)}_{2} + \underbrace{w(A, B)}_{3} & = 5 \end{cases} = 5$$

$$dist(C) = \min \begin{cases} dist(C) & \infty \\ \underbrace{dist(A)}_{2} + \underbrace{w(A, C)}_{4} & = 6 \end{cases}$$

$$dist(C) = \min \begin{cases} dist(C) = 6 \\ \underbrace{dist(B)}_{5} + \underbrace{w(B, C)}_{5} = 10 \end{cases}$$

10/16/23

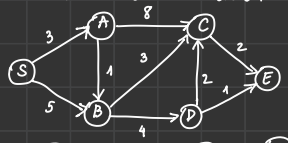
Shortest path on DAG:

- sort vertices in topological order
- perform relaxation on the ordered nodes.

★ DAG: topological order using DFS !!!

★ TC: O(|V| + |E|)

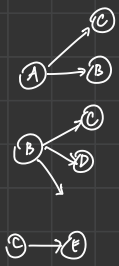
ex



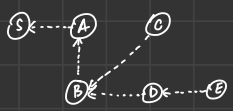
dist

S	A	B	D	C	E
0	~~∞~~ 3	~~∞~~ ~~8~~ 4	~~∞~~ 8	~~∞~~ ~~11~~ 7	~~∞~~ 9

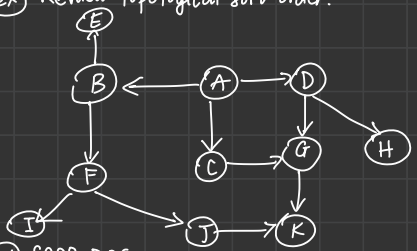
Relax edges



Shortest path tree



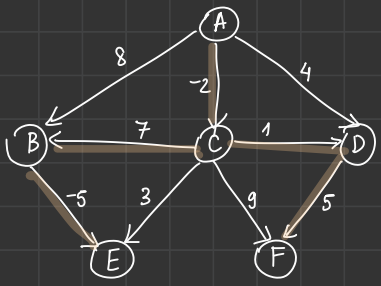
ex Review topological sort order:



dependency first

A D H C G B E F J K I

ex SSSP DAG:



directed + acyclic ✓ ⇒ sequence

★ use DFS to find topological order : A → C → B → D → F → E

→ relax all edges outgoing edges from that node

A	C	B	D	F	E
0	~~∞~~ -2	~~∞~~ ~~8~~ 5	~~∞~~ ~~4~~ -1	~~∞~~ ~~7~~ 4	~~∞~~ ~~1~~ 0

↑

10/17/23

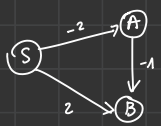
Shortest distance problem: Dijkstra's algorithm

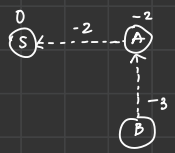
on general directed graphs, without negative weights

Input: $G(V,E)$, edge length $\ell: E \to R_{\geq 0}$, and $s \in V$

Output: $\forall v \in V$, smallest distance $dist(u)$ from s to u.

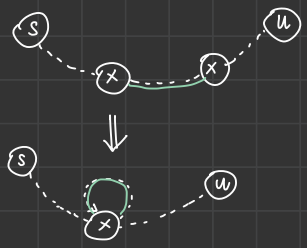
$O(|\log|V| \times |V| + |E|)$
↓
dequeue



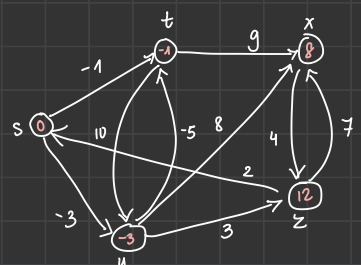


Shortest path $dist(s,v) = d^*(u)$

1) Without negative cycle in the graph the vertices on the graph $s \rightsquigarrow v$ are unique
2) # edges on the path $s \rightsquigarrow v \leq n-1$
3) After all edges on the path $s \rightsquigarrow u$ are relaxed, $dist(s,u) = d^*(u)$



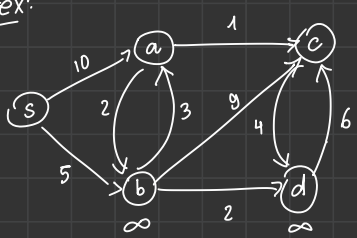
ex:



$\Rightarrow$

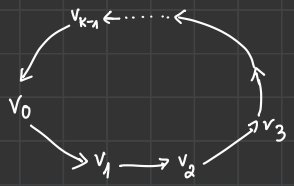
V =	s	t	y	x	z
K=1	0	∞	∞	∞	∞
K=2	0	-1	-3	8	∞ 12

ex:



↓

s	a	b	c	d
0	∞ 10 8	∞ 5	∞ 14 13 9	∞ 7



$$dist(v_0) + \ell(v_0, v_1) \geq dist(v_1)$$
$$dist(v_1) + \ell(v_1, v_2) \geq dist(v_2)$$
$$\vdots$$
$$dist(v_{k-1}) + \ell(v_{k-1}, v_k) \geq dist(v_k)$$

$$\sum_{i=0}^{k-1} \ell(v_i, v_{i+1}) \geq 0$$

contradict

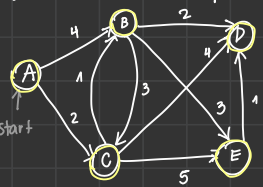
$$\sum_{i=0}^{k-1} (v_i, v_{i\,k+1}) < 0$$

★ YOUTUBE LEARNING

★ Dijkstra's: 2 main steps — i) update estimates; ii) choose a vertex

doesn't work on negative edges!

ex



A	0
B	~~∞~~ ~~4~~ 3
C	~~∞~~ 2
D	~~∞~~ ~~6~~ 5
E	~~∞~~ ~~7~~ 6

no updates to table if the vertex being examined has no edges leaving

Unvisited: {A, B, C, D, E}

Time complexity: $O(|E| + |V| \log |V|)$ (Fibonacci heap is used)

Quick Algorithm Overview

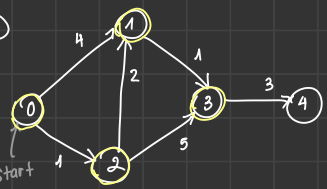
Maintain a 'dist' array where the distance to every node is positive infinity. Mark the distance to the start node 's' to be 0.

Maintain a PQ of key-value pairs of (node index, distance) pairs which tell you which node to visit next based on sorted min value.

Insert (s, 0) into the PQ and loop while PQ is not empty pulling out the next most promising (node index, distance) pair.

Iterate over all edges outwards from the current node and relax each edge appending a new (node index, distance) key-value pair to the PQ for every relaxation.

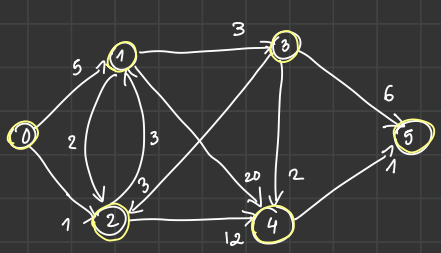
ex



dist

0	1	2	3	4
0	~~∞~~ ~~4~~ 3	~~∞~~ 1	~~∞~~ ~~6~~ 4	~~∞~~ 7

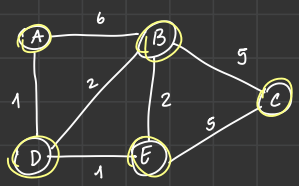
ex



dist

0	1	2	3	4	5
0	~~∞~~ ~~5~~ 4	~~∞~~ 1	~~∞~~ 7	~~∞~~ 13 9	~~∞~~ 13 10

ex



dist

A	B	C	D	E
0	~~∞~~ ~~6~~ 3	~~∞~~ 7	~~∞~~ 1	~~∞~~ 2

10/17/23

Short

Bellman-Ford Algorithm (negative edges :) → single-source shortest path

- For graphs that may contain negative edges, edge relaxation cannot be done in specific order.

⇒ But every shortest path consists of at most n-1 edges; n-1 rounds of edge relaxation suffices

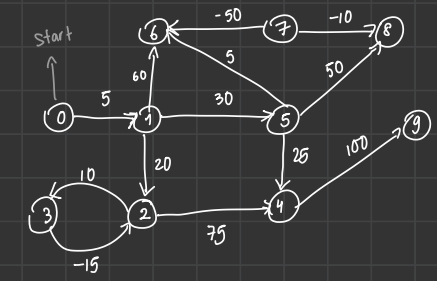
TC: O(|V||E|) not recommended! When to use? negative edges ⇒ BF can detect negative cycles !!! :)

Why does Dijkstra's fail on negative edges → infinite loop b/c alg keeps finding better and better path

dist

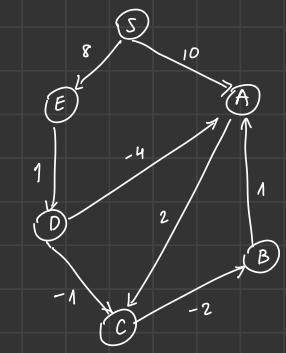
Theory: ✓ negative cycles

iterates at most n-1 iterations



0	0
1	~~∞~~ 5
2	~~∞~~ ~~28~~ 20
3	~~∞~~ 35
4	~~∞~~ ~~100~~ 60
5	~~∞~~ 35
6	~~∞~~ ~~68~~ 40
7	~~∞~~ -10
8	~~∞~~ ~~85~~ -20
9	~~∞~~ 200

ex



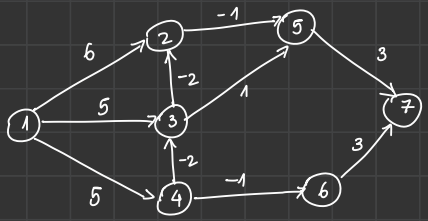
S	A	B	C	D	E
0	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~
	~~10~~	~~10~~	~~12~~	9	8
	5	5	~~8~~ 7		

max 5 iterations

. . . . no change in value

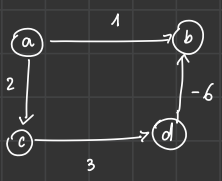
algo takes at most n-1 iterations

ex



1	2	3	4	5	6	7
0	∞	∞	∞	∞	∞	∞
0	6	5	5	∞	∞	∞
0	3					

ex



order of edges to relax: (d, b), (c, d), (a, c), (a, b)

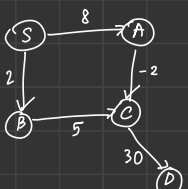
↓ start a	b	c	d	round of relaxation
0	∞	∞	∞	
0	1	2	∞	1st
0	1	2	5	2nd
0	-1	2	5	3rd → answer !!!

10/19/23

Quiz on Tues Oct 24:

- Strongly connected components
- Single shortest path on DAG
- _____ on Dijasktra's
- _____ Bellman-Ford

ex:



dist

S	A	B	C	D
0	~~∞~~	~~∞~~	~~∞~~	~~∞~~
	8	2	7	10

Review of strongly connected components:

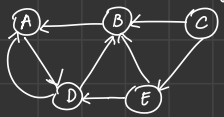
- DFS on G
- generate G^T
- DFS on G^T with vertex v with highest post(v) value.

Quiz 5 Review:

Review of strongly connected components:

- DFS on G
- generate G^T
- DFS on G^T with vertex v with highest post(v) value.

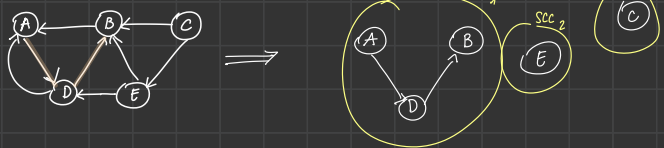
(1) Draw the transposed graph:



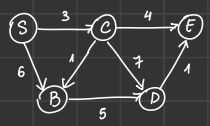
order for step 3 from high to low post time stamp:

A, B, D, E, C

Draw the 2nd DFS forest, and based upon which, to identify all SCCs



(2) Run shortest-path DAG on given DAG. Show dist values for all vertices after every edge relaxation in a topological order including initial values.



Topological order: S B C D E

start

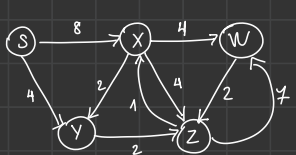
S	B	C	D	E	relaxed edge
0	∞	∞	∞	∞	
0	∞	(3)	∞	∞	(S,C)
0	(6)	3	∞	∞	(S,B)
0	(4)	3	∞	∞	(C,B)
0	4	3	(10)	∞	(C,D)
0	4	3	10	(7)	(C,E)
0	4	3	(9)	7	(B,D)
0	4	3	9	7	(D,E)

can relax B or C first, either order is fine — and also order of relaxation can be diff

DAG: Topological sort -> then relaxation

$O(V+E)$

(3) Run dijskra's. Show dist values for all vertices after every vertex is dequeued and relavant edges are relaxed



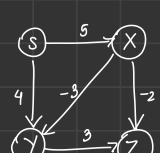
S	X	Y	Z	W	dequeued vertex
0	∞	∞	∞	∞	
0	(8)	(4)	∞	∞	S
0	8	4	(6)	∞	Y
0	(7)	4	6	(13)	Z
0	7	4	6	(11)	X
0	7	4	6	11	W

Choose best unvisited node

TC: $O(E + V \log V)$

priority queue: (node, index) distance pair

(4) Run Bellman Ford algorithm on this graph w/ negative edges. Assume at each round of relaxation, the edges are relaxed in order of (X,Z), (Y,Z), (S,X), (S,Y), (X,Y). Show dist values for all vertices.



S	X	Y	Z	round of relaxation
0	∞	∞	∞	
0	5	2	∞	1
0	5	2	3	2
0	5	2	3	3
0	5	2	3	4

Chap 4: Dynamic Programming

$$Fibonacci = \begin{cases} 1 & n = 1,2 \\ Fib(n-1) + Fib(n-2) & n \geq 3 \end{cases}$$

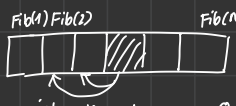
$\Rightarrow \sqrt{2}^N \leq T(n) \leq 2^N$ 1.6180^N

not good in terms of running time

⇒ looking for something efficient

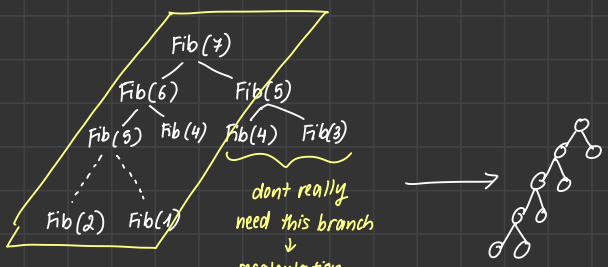
① Naive recursive algorithm:

② Fib(1) Fib(2) ... Fib(N)



interative alg ... $O(n)$

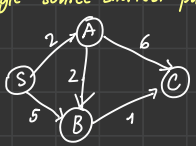
③ recursion ??



* Intuitialize $F(K) = -1, \forall K$

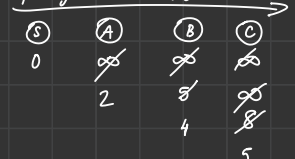
```
func memorized_Fib(N)
    if N = 1 or N = 2
        F(N) = 1;
    else
        if F(n-1) == -1
            call memorized_Fib(N-1)
        if F(N-2) == -1
            call.... Fib(N-2)
        F[N] = F[N-1] + F[N-2]
```

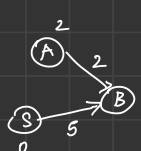
Problem 1: Single-source shortest paths in DAG



DAG

Topological sort order



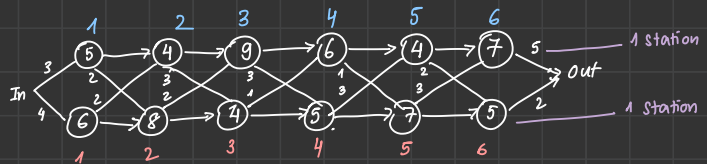


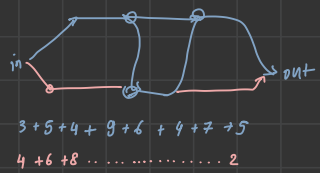
DP

$$dist(B) = \min \begin{cases} dist(A) + w(A,B) \\ dist(S) + w(S,B) \end{cases}$$

Problem 2: Assembly line rescheduling

The fatest path through a factory

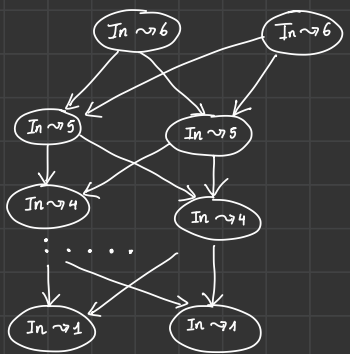




- 2n station, each station has processing time
- no time cost for transitions within the same production line
- a path time = sum of all processing and transition times on the path

output: a path through the factory of a min time

path: (1 2 1 1 2 1) (0 1 0 0 1 0)



path time

Step 1 Analyze the problem

The fastest path In ⇝ Out has to be:

the faster of { fastest path In ⇝ 6 then edge 6 → Out
fastest path In ⇝ 6 then edge 6 → Out

★ In general:

- for every K = 2, 3, ..., n

the fastest path In ⇝ K has to be

the faster of { a fastest path In ⇝ k-1 then edge k-1 → k
a fastest path In ⇝ k-1 then edge k-1 → k

the fastest path In ⇝ K has to be

the faster of { a fastest path In ⇝ k-1 then edge k-1 → k
a fastest path In ⇝ k-1 then edge k-1 → k

Base:

fastest path In ⇝ 1 is In ⇝ 1

In ⇝ 1 is In → 1

Step 2: Define numerical objective function

For $k = 1, 2, \ldots, n$ (station #); $i = 1, 2$ (production line), label w/ (1,1) ... (1,n) for stations in production line 1, and with (2,1) ... (2,n) for production line 2.

Let $pt_i(k)$ be the processing time on station (i,k)

Let $tt_i(k-1)$ be transfer time from station $(i, k-1)$ to station $(\bar{i}, k)$, where $\bar{i}$ is the opposite prod. line of i.

function $ft_i(K)$: fastest of a path from station In to station (i,k)

Then $ft_i(K) = \min \begin{cases} ft_i(K-1) + pt_i(K) \\ ft_{\bar{i}}(K-1) + tt_{\bar{i}}(K-1) + pt_i(K) \end{cases}$ $K \geq 2$

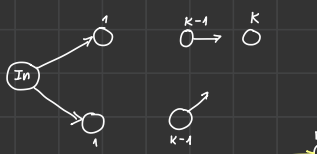
$ft(1)$ = the known time from I to station $(i,1) + pt_i(1)$

Step 3 Establish & fill DB tables $i = 1,2 ; k = 1,2,\dots n$

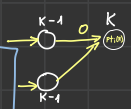
- establish a table $F_{2 \times n}$ to store values of function $ft_i(k)$
- establish a table $prev_{2 \times n}$ to store previous stations
- fill the tables using recursive formulas for $ft_i(k)$, w/ an iterative program.

10/24/23

DP = non naive exhaustive search + overlapping subproblems.



fastest time
fastest path



$$ft_1(K) = Pt_1(K) + \min \begin{cases} ft_1(K-1) + 0 \\ ft_2(K-1) + tt_2(K-1) \end{cases}$$

$$ft_2(K) = Pt_2(K) + \min \begin{cases} ft_1(K-1) + tt_1(K-1) \\ ft_2(K-1) + 0 \end{cases}$$

ft: fastest time
pt: processing time
tt: transition time

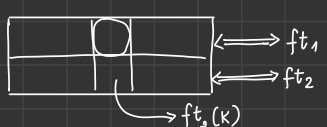
In $\rightsquigarrow$ (2,K-1)

n x 2

Base: $K = 1$

$ft_1(1) = Pt_1(1) + tt_{In,1}$

$ft_2(1) = Pt_2(1) + tt_{In,2}$



$F[2,K] = ft_2(K)$

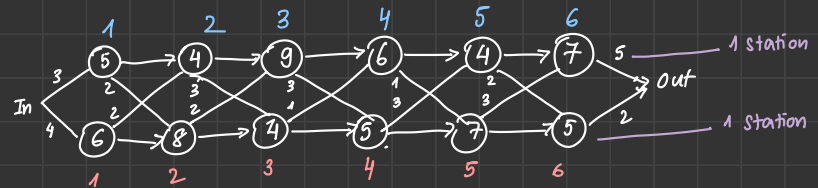
```
function fastest path (pt, tt, n)
    F[1, 1] = Pt1(1) + tt In,1;
    F[2, 1] = Pt2(1) + tt In,2;
    For K = 2 to n:
        A = F[1, K-1] + Pt1(K)
        B = F[2, K-1] + Pt1(K) + tt2(K-1)
        if A ≥ B
            F[1, K] = B;
            prev[1, K] = (2, K-1)
        else F[1, K] = A
             prev[1, K] = (1, K-1)
```

prev

Step 4: Trace back fastest path

- from the fastest time, we know the last station of which production line is on the fast path before station Out.

10/25/23



	1	2	3	4	5	6
1	3+5	12	21	26	30	37
2	4+6	18	19	24	31	36

In → row 1 col 1; out → row 2 col 6

$38 = \min\{42, 38\}$

$F[i,k]$

$Prev[i,k]$

$F[2,3] = 19$

$j = Prev[2,3] = 1$

$j = 1$

$(1,2) \rightarrow (2,3)$

$prev(out) = 2$

function TracePath (i, K)

if k = 1 printout

else

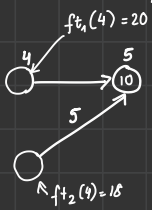
j = prev (i, k);

Tracepath (j, k-1)

print (edge (j, k-1) → (i, k))

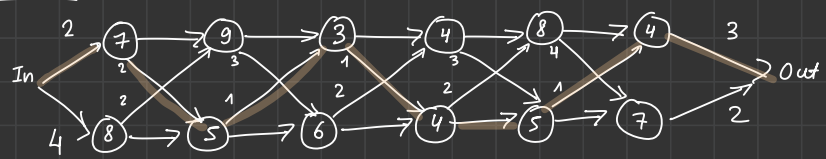
How to fill in DP table?

ex)



$ft_1(5) = \min\{ft_1(4) + 0 + Pt_1(5) = 20 + 0 + 10 = 30,\ ft_2(4) + tt_2(4) + Pt_1(5) = 18 + 5 + 10 = 33\}$

Another ex:



	1	2	3	4	5	6
$f_1[i]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

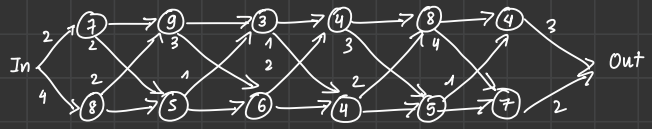
$\min\{38, 39\}$

$\min\{18, 23\}$

$f_2[2] = \min\{16, 17\}$

$f_1[3] = \min\{21, 20\}$

ex: Assembly line scheduling



$$ft_1(K) = \min \begin{cases} ft_1(K-1) + pt_1(K) \\ ft_2(K-1) + tt_2(K-1) + pt_1(K) \end{cases}$$

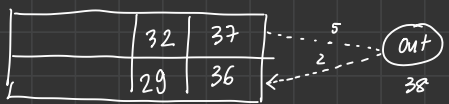
$$ft_2(K) = \min \begin{cases} ft_1(K-1) + tt_1(K-1) + pt_2(K) \\ ft_2(K-1) + pt_2(K) \end{cases}$$

Also tracing our way thru the factory:

- Keep track of the stations we passed through in the fastest route
- Let $L_i[j]$ be the line number, either 1 or 2, whose stations $j-1$ is used in the fastest way thru station S_{ij}.
- Can either trace path directly w/ arrows like prev page, or have a L table.

j	2	3	4	5	6	
$L_1[j]$	1	2	1	1	2	$L^* = 1$
$L_2[j]$	1	2	1	2	2	

10/26/23



G. s.t

? $s \rightsquigarrow t$

Input: X

Output: Y

$|Y| = 1$ ⟸ decision problem

$|Y| \geq 2$ ⟸ search problem

Optimization problem

For ex: fastest factory path

max/min value.

Tracebacks take less time than filling the table itself.

* Characteristics of problems that can be solved with DP:

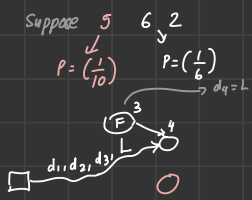
1) optimal substructures

Problem 3: Decoding dishonest dice rolls.

ex: Decoding dishonest dice rolls

$O = o_1 o_2 \dots o_n$ observed dice roll outcomes.

$S = d_1 d_2 \dots d_n$ the sequence of dice w/ highest probability



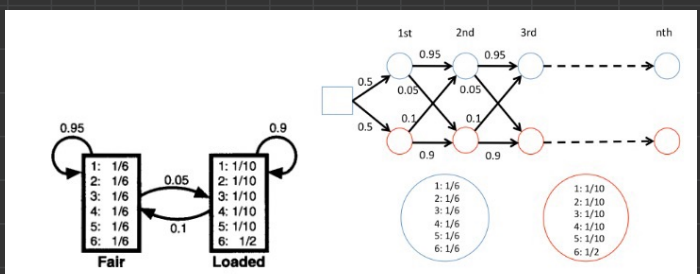
d_1, d_2, d_3, d_4 L

$= \max$ { ..... F → F ; ..... L }

d_1, d_2, d_3, d_4 L

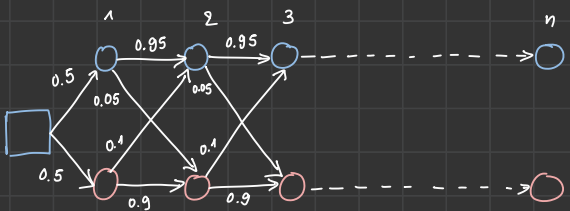
$= \max$ { ..... F → L ; ..... L → L }

find most likely path (path w/ highest probability)



$O = o_1 o_2 \dots o_n$ observed dice roll outcomes;

$S = d_1 d_2 \dots d_n$ the sequence of dice **with highest probability**



- emission probability $e_{F(k)} = \frac{1}{6}$ for all $k = 1, 2, \dots, 6$
- transition probability:

$$t_{FF} = 0.95,\ t_{FL} = 0.05,\ t_{LL} = 0.9,\ t_{LF} = 0.1$$

- compute probability of rolling 2466 with dice FFLL

$$0.5 \times e_F(2) \times t_{FF} \times e_F(4) \times t_{FL} \times e_L(6) \times t_{LL} \times e_L(6)$$

Probability of dice rollings:

- emission probability $e_F(k) = \frac{1}{6}$

ex: 2 4 6 6

F F L L

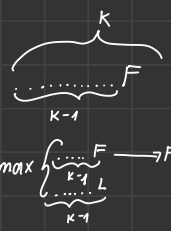
$0.5 \times \frac{1}{6} \times 0.5 \times \frac{1}{6}$

Step 1: Problem analysis

*It's anology between decoding dice and finding the fastest path through factory:

- sequence of dice consists of either F or L dices in each position = a path thru factory consists of stations either in line 1 or 2.
- the most like sequence is the one w/ highest probability = the fastest one is smallest time.

- most likely sequence ends at either fair or loaded dice.
- for $K \geq 1$, the most likely sequence of length K ending at fair dice is.



whichever has the highest probability:

1) either the most likely sequence of length K-1 ending at fair die followed by fair die
2) or the most likely sequence of length K-1 ending at loaded die followed by fair die

Step 2: Define objective function

Define $m(k, F)$ highest probability of a sequence of k dice ending at Fair die to emit the first k observed number.

$$m(k,F) = \max \begin{cases} m(k-1,F) \times t_{F,F} \times e_F(o_k) \\ m(k-1,L) \times t_{L,F} \times e_F(o_k) \end{cases} = \max \begin{cases} \text{K-1: F} \to F \\ \text{K-1: L} \to F \end{cases}$$

$$m(k,L) = \max \begin{cases} m(k-1,F) \times t_{F,L} \times e_L(o_k) \\ m(k-1,L) \times t_{L,L} \times e_L(o_k) \end{cases}$$

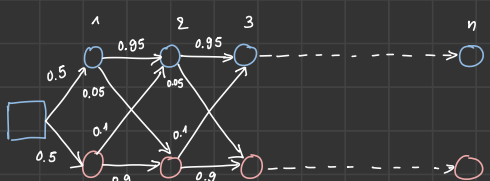
$m(k, L)$

= .....

0 = 2 4 6 6

$m(1,F) = 0.5 \times e_F(o_1)$

$m(1,L) = 0.5 \times e_L(o_1)$



$t_{FF} = 0.95$, $t_{FL} = 0.05$, $t_{LL} = 0.9$, $t_{LF} = 0.1$

Step 3 Fill DP tables

0 = 2 4 6 6

max of 2

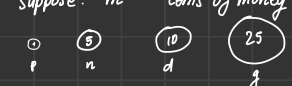
$0.5 \times \frac{1}{6} = 0.08$	$\max \begin{cases} 0.08 \times 0.95 \times \frac{1}{6} = \boxed{0.01216} \\ 0.05 \times 0.1 \times \frac{1}{6} = 0.00083 \end{cases}$	$\max \begin{cases} 0.01216 \times 0.95 \times \frac{1}{6} = \boxed{0.001925} \\ 0.0045 \times 0.1 \times \frac{1}{6} = 0.000075 \end{cases}$	$\max \begin{cases} 0.001925 \times 0.95 \times \frac{1}{6} = \boxed{0.00030479} \\ 0.002025 \times 0.1 \times \frac{1}{6} = 0.00003375 \end{cases}$	F
$0.5 \times \frac{1}{10} = 0.05$	$\max \begin{cases} 0.08 \times 0.05 \times \frac{1}{10} = 0.004 \\ 0.05 \times 0.9 \times \frac{1}{10} = \boxed{0.0045} \end{cases}$	$\max \begin{cases} 0.01216 \times 0.05 \times \frac{1}{2} = 0.000292 \\ 0.0045 \times 0.9 \times \frac{1}{2} = \boxed{0.002025} \end{cases}$	$\max \begin{cases} 0.001925 \times 0.05 \times \frac{1}{2} = 0.0000481 \\ 0.002025 \times 0.9 \times \frac{1}{2} = \boxed{0.00091125} \end{cases}$	L
2	4	6	6	

Problem 4: Knapsack problem

- Input: n items, of size s_i and value v_i, $i = 1, \dots, n$ and knapsack of volume W
- Output: a subset of items $A \subseteq \{1, 2, \dots, n\}$ such that $\sum_{i \in A} v_i$ is maximized, subject to $\sum_{i \in A} s_i \leq W$

aka coin change

Suppose: m — cents of money



let $c(m)$ = minimum number of coins to change m cents

$$c(m) = \min \begin{cases} c(m-25)+1 & m \geq 25 \\ c(m-10)+1 & m \geq 10 \\ c(m-5)+1 & m \geq 5 \\ c(m-1)+1 & m \geq 1 \end{cases}$$

Base: $c(0) = 0$

0-1 knapsack | fractional

	1	2	3	.....	n
size	s_1	s_2	s_3	.....	s_n
value	v_1	v_2	v_3	.....	v_n

{ pick item n / not pick item n

size $W \Rightarrow W - s_n$

value $+ v_n$

Knapsack analysis:

- how to select some items from the first k items into a space of ?

volume X, $X \leq W$.

- either item k is selected, with gain of value v_k but decrease of available space to $X - s_k$.
- or discard item k, with no change in value and no change in available space.

Def: $V(W, n)$: maximum value to pack some items from $\{1, 2, \dots n\}$ into the knapsack of size W.

$$V(w, n) = \max \begin{cases} V(w - s_n, n-1) + v_n \\ V(w, n-1) + 0 \rightarrow \text{didn't pack anything} \end{cases} \quad w \geq s_n$$

($V(w,n)$: size w, item #)

Base: $V(w, 0) = 0$

$V(0, n) = 0$

10/30/23

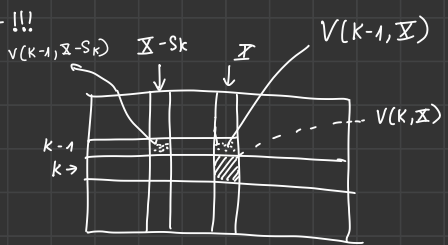
$W = 8$

	1	2	3
S	3	5	2
V	17	15	20

move to top ↑, abandon this item 2

NO SORTING !!!

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	17	17	17	17	17	17
2	0	0	0	17	17	17			
3									



$V(K, X)$ → allowed space

↑ represents the first K items.

max value of selected items from $\{1, 2, \dots k\}$ that can be packed into space X.

$V(1,2) = \max \{$

$V(1,3) = \max \begin{cases} V(1-1, 3-3) + 17 \\ V(1-1, 3) = 0 \end{cases}$

$V(1,4) = \max \begin{cases} V(1-1, 4-3) + 17 \\ V(1-1, 4) \end{cases}$

$V(2,3) = \max \begin{cases} V(2-1, 3-3) + 15 \\ V(2-1, 3) = V(1,3) = 17 \end{cases}$

$V(2,5) = \max \begin{cases} V(2-1, 5-5) + 15 \\ V(2-1, 5) = V(1,5) = 17 \end{cases}$

KNAPSACK YOUTUBE:

DP table to store states:

	0	1	2	3	4	5	6	7
empty	0	0	0	0	0	0	0	0
$v_1=2, w_1=3$	0	0	0	2	2	2	2	2
$v_2=2, w_2=1$	0	2	2	2	4	4	4	4
$v_3=4, w_3=3$	0	2	2	4	6	6	6	8
$v_4=5, w_4=4$	0	2	2	4	6	7	7	9
$v_5=3, w_5=2$	0	2	3	5	6	7	9	10

items (rows 1–5)

compare each cell w/ this capacity

best value

Knapsack capacity

a cell is a value

* don't include current item: look 1 row above ↑
* include current item: look 1 row above but shift over by weight of current item + add current value

$$V(K,X) = \max \begin{cases} V(K-1, X-s_k) + V_k \\ V(K-1, X) \end{cases}$$

Another way by using detailed calculations:

	0	1	2	3	4	5	6	7
empty	0	0	0	0	0	0	0	0
$v_1=2, w_1=3$	0	0	0	2	2	2	2	2
$v_2=2, w_2=1$	0	2	2	2	4	4	4	4
$v_3=4, w_3=3$	0	2	2	4	6	6	6	8
$v_4=5, w_4=4$	0	2	2	4	6	7	7	9
$v_5=3, w_5=2$	0	2	3	5	6	7	9	10

Knapsack capacity

$V(1,1) = \max \begin{cases} V(0,-2) + 2 = -\infty \\ 0 \end{cases} = 0$

$V(1,3) = \max \begin{cases} V(0,0) + 2 \\ 0 \end{cases} = 2$

$V(2,1) = \max \begin{cases} V(1,0) + 2 = 0 + 2 = 2 \\ V(1,1) = 0 \end{cases}$

$V(2,3) = \max \begin{cases} V(1,2) + 2 = 2 \\ V(1,3) = 2 \end{cases}$

$V(2,4) = \max \begin{cases} V(1,3) + 2 = 4 \\ V(1,4) = 2 \end{cases}$

to keep track of what items to store

we selected 3 out of 5: item 2, item 4, item 5

if current is diff from above, add it to bag and move diagonally by its weight

Notes:

↑: not looking at item

+v: include item

* Running time of Knapsack: $O(nW)$

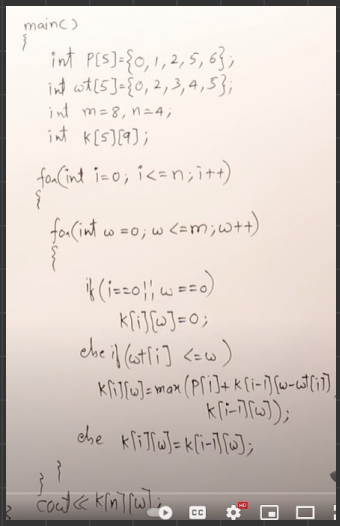
ex 0/1 knapsack:

	1	2	3	4
V	1	2	5	6
W	2	3	4	5

Bag capacity: 8

n = 4

	0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0
v=1, w=2	1	0	0	1	1	1	1	1	1
v=2, w=3	2	0	0	1	2	2	3	3	3
v=5, w=4	3	0	0	1	2	5	5	6	7	7
v=6, w=5	4	0	0	1	2	5	6	6	7	8



```
main()
{
    int P[5]={0,1,2,5,6};
    int wt[5]={0,2,3,4,5};
    int m=8, n=4;
    int k[5][9];

for(int i=0; i<=n; i++)
{
    for(int w=0; w<=m; w++)
    {
        if(i==0 || w==0)
            k[i][w]=0;
        else if(wt[i] <=w)
            k[i][w]=max(P[i]+k[i-1][w-wt[i]]
                        k[i-1][w]);
        else k[i][w]=k[i-1][w];
    }
}
cout<<k[n][w];
```

EDIT DISTANCE YOUTUBE:

- measuring distance between 2 input strings, based on how many
  1) matches
  2) insertions
  3) deletions
  4) mismatches

For 2 strings: X of length n
Y _____ m
⇒ edit distance of X & Y is D(n, m)

⇒ D(i,j): edit distance between X[1...i] and Y[1...j]

★ DP for edit distance:

- A tabular computation of D(n,m)
- Solving problems by combining solutions to subproblems
- Bottom up: _ compute D(i,j) for small i, j.
  _ compute larger D(i,j) based on prev computed smaller values
  _ compute D(i,j) for all i (0<i<n) and j (0<j<m)
- Initialization: D(i,0) = i
  D(0,j) = j
- Recurrence relation: For each i = 1...M
  For each j = 1...N

$$D(i,j) = \min \begin{cases} D(i-1,j)+1 & \text{deletion } 1 \\ D(i,j-1)+1 & \text{insertion } 1 \\ D(i-1,j-1)+2 \begin{cases} \text{if } x(i) \neq Y(j) & \text{mismatch/sub } 2 \\ \text{if } x(i) = Y(j) & \text{match } 0 \end{cases} \end{cases}$$

Termination: D(n,m) is the edit distance

★ Fill DP table: from top to bottom
left to right

result

N	9	8	9	10	11	12	11	10	9	(8)
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{deletion } 1 \\ D(i,j-1) + 1 & \text{insertion } 1 \\ D(i-1,j-1) + 2 \begin{cases} \text{if } X(i) \neq Y(j) & \text{mismatch/sub } 2 \\ \text{if } X(i) = Y(j) & \text{match } 0 \end{cases} \end{cases}$$

???

replace cost here is 2!

$\min \begin{cases} 2+1 \\ 3 \\ 3 \end{cases}$

$\min \begin{cases} 1+1 \\ 1+1 \\ 0+2 \end{cases} = 2$

Another example But 3 operations all cost 1

ex:

		0	1	2	3	4	5	6	7
		"	r	e	p	l	a	c	e
0	"	0	1	2	3	4	5	6	7
1	d	1	1	2	3	4	5	6	7
2	e	2	2	1	2	3	4	5	6
3	l	3	3	2	2	2	3	4	5
4	e	4	4	3	3	3	3	4	4
5	t	5	5	4	4	4	4	4	5
6	e	6	6	5	5	5	5	5	4

first row

replace	delete
insert	(i, j)

(i, j) → current position

Notes:

insert → replace

remove

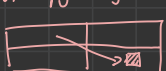
if $X[i] \neq Y[j]$

replace	remove
insert	min (replace, remove, insert) + 1

→ or other cost of mismatch

If $X[i] = Y[j]$

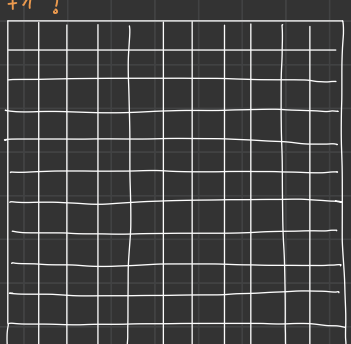
just copy diagonal element



ex:

		""	a	b	c	f	g
		0	1	2	3	4	5
a		1	0	1	2	3	4
d		2	1	1	2	3	4
c		3	2	2	1	2	3
e		4	3	3	2	2	3
0		5	4	4	3	3	2

replace cost is +1 !



☆ Coin Change DP problem:

$d_1: 1 \quad d_2: 5 \quad d_3 = 10 \quad d_4 = 25$

goal: smallest # of coins for change

- $C[p]$: minimum number of coins needed to make change for p cents.
- x: value of the first coin used in optimal solution

$$C[p] = 1 + C[p-x]$$

(ex) Just penny, nickel, dime

$$C[p] = 1 + \min \begin{cases} C[p-1] \\ C[p-5] \\ C[p-10] \end{cases}$$

$C[0] = 0$

(ex) $C[1] = \min \begin{cases} 1 + C[1-50] = \infty \\ 1 + C[1-25] = \infty \\ 1 + C[1-10] = \infty \\ 1 + C[1-1] = 1 \end{cases}$

$C[2] = \begin{cases} 1 + C[2-50] = \infty \\ 1 + C[2-25] = \infty \\ 1 + C[2-10] = \infty \\ 1 + C[2-1] = 2 \end{cases}$ ... similarly, $C[3] = 3 \ldots$ $C[10] = 1$

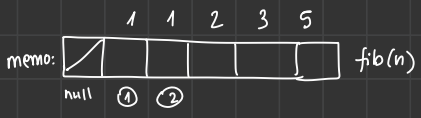
$$C[11] = \min \begin{cases} 1 + C[11-50] = \infty \\ 1 + C[11-25] = \infty \\ 1 + C[11-10] = 2 \\ 1 + C[11-1] = 2 \end{cases}$$

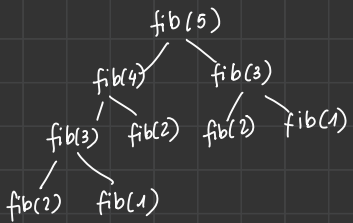
$$\text{Coins}[i] = 1 + \min_{d_j \leq i} \{\text{coins}[i-d_j]\}$$

YOUTUBE REVIEW OF DYNAMIC PROGRAMMING

1. Recursion
2. Store (Memoize)
3. Bottom-up

1, 1, 2, 3, 5, 8, ...






```
def fib(n, memo):                    1...n
    if memo[n] != null:              n
        return memo[n]
    if n == 1 or n == 2:             n
        result = 1
    else:
        result = fib(n-1) + fib(n-2)
        memo[n] = result
    return result
```

T(n) # calls to func fib

$T(n) \leq 2n + 1$

$T(n) = O(n)$

Practice:

① Longest increasing subsequence:

LIS([5 2 8 6 3 6 9 5]) len = 4

10/31/23

Knapsack

$W = 8$

	1	2	3
S	3	5	2
V	17	15	20

i \ k	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	17	17	17	17	17	17
2	0					17			
3	0								

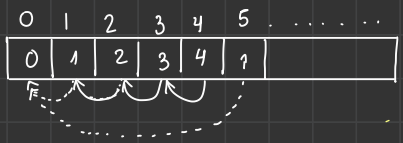
$$V(2,5) = \max \begin{cases} V(1,0) + 15 \\ V(1,5) \end{cases}$$

(ex) cents of money

$$c(m) = \min \begin{cases} c(m-25)+1, & m \geq 25 \\ c(m-10)+1, & m \geq 10 \\ c(m-5)+1, & m \geq 5 \\ c(m-1)+1, & m \geq 1 \end{cases}$$

$c(0) = 0$

$$c(5) = \min \begin{cases} c(0)+1 \\ c(4)+1 \end{cases}$$

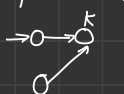


---

minimum edit distance

① longest common subsequence

② pairwise alignment



exhaustive search (enumeration)

1 2 . . . . (n)

1 2 . . . . (m)

① 1 . . . . n−1

1 . . . . m−1

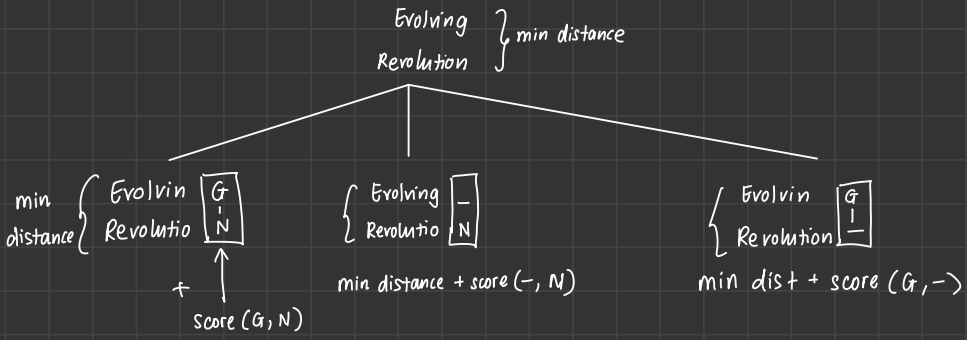
② 1 . . . . n

1 . . . . m−1

③ 1 . . . . n−1

1 . . . . m

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$



⇒ 3 possible scenarios

⇒ 3 subproblems

✱ Define: input 2 strings $x[1 \dots m]$ and $y[1 \dots n]$

$E(i,j)$ smallest dist (lowest score between prefixes $x[1 \dots i]$ and $y[1 \dots j]$

$$E(i,j) = \min \begin{cases} E(i-1,j-1) + \text{score}(X[i], Y[j]) \\ E(i,j-1) + \text{score}('\_', Y[j]) \\ E(i-1,j) + \text{score}(x[i], '\_') \end{cases}$$

$E(0,0)$

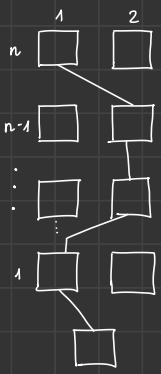
$E(0,3) = 3$

$E(i,0)$

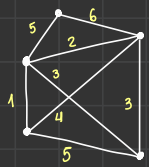
✱ $$E(i,j) = \min \begin{cases} E(i-1,j-1) + \text{diff}(i,j) \\ E(i,j-1) + 1 \\ E(i-1,j) + 1 \end{cases}$$ where $\text{diff}(i,j) = \begin{cases} 0 & x[i] = y[i] \\ 2 & x[i] \neq y[i] \end{cases}$

greedy algorithms

11/02/23



- A spanning tree is a spanning subgraph that is a tree.
- A minimum spanning tree is a spanning tree of the smallest total edge weights.



Let $G = (V, E)$
A subgraph $H = (U, F)$ of G is a spanning subgraph if $V = E$

11/06/23

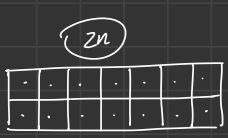
Let $H \leq G$ def: $mst(H)$ to be the min cost of a spanning tree in H.

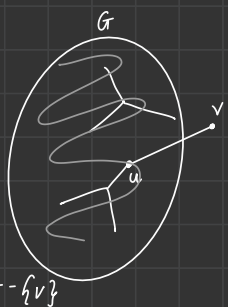
$$mst(G) = \min_{v \in V} \{ mst(G - \{v\}) + \min_{u \in V - \{u\}} W(v,u) \}$$

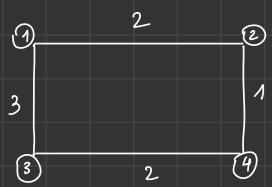
$G = (V, E)$

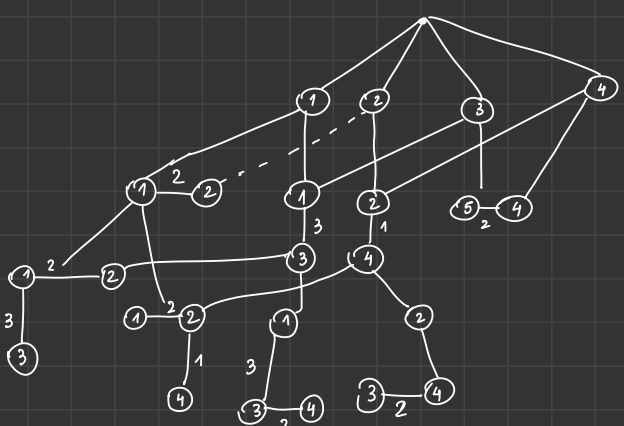
$\forall H \subseteq G$

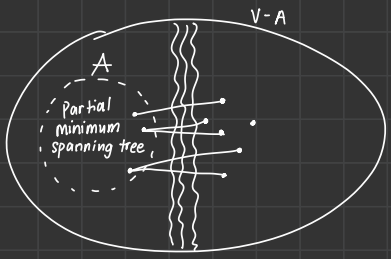
$u \in N(v)$
s.t. $w(u,v)$ is smallest





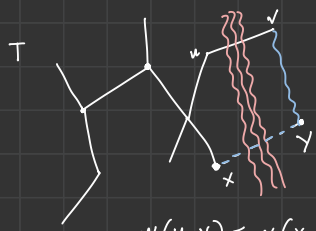






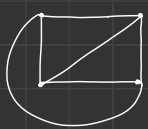
a cut separates a graph

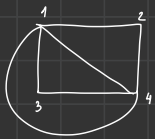
$A \neq \emptyset$

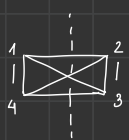


$w(u,v) \leq w(x,y)$

$w(T') \leq w(T)$







$S = \{3,2\}$ $V-S = \{1,4\}$

11/07/23

2. Greedy algorithms

Theorem: A greedy-choice property for MST problem:

Let G be a given graph. Then any light edge crossing any cut of the graph is in some minimum spanning tree of the graph.

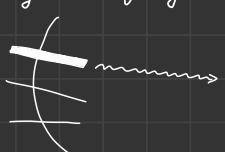
Proof: (using the Exchange method)

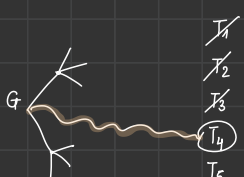
- let T be an m.s.t. for G and (u,v) is a light edge crossing some cut $(S, V-S)$;
- if (u,v) is in T, then the theorem is proved;
- otherwise, let edge (x,y) in T that crosses the cut $(S, V-S)$;
- then $T \cup \{(u,v)\}$ contains a cycle; why?
- let $T' = T \cup \{(u,v)\} - \{(x,y)\}$. Then T' is a spanning tree; why
- because (x,y) and (u,v) cross $(S, V-S)$ and (u,v) is a light edge, T' is also m.s.t. for G. why?
- Because T' contains (u,v), the theorem is proved.

# of cuts in G? $(S, V-S)$ $S \leq V$

2^n

Running time = height of tree. Usually greedy algorithms give linear time.

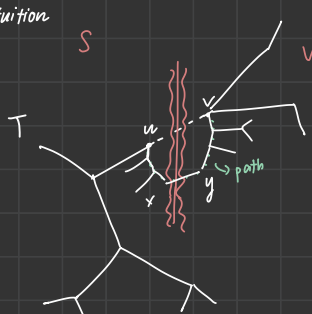




Proof:

* Intuition

crosses a cut



Let (u,v) be a light edge cross some cut $(s, v-s)$

Let T be an arbitrary MST:

① If $(u,v) \in T$, done! ✓

② Otherwise, $\exists (x,y)$ in T cross the cut

$w(x,y) \geq w(u,v)$

Create $T' = T - \{(x,y)\} \cup \{(u,v)\}$

T' is mst & it contains (u,v) ⟸ Claim: T' is a spanning tree

$w(T') \leq w(T)$

MST at case 2

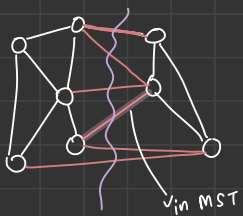
In order for $T = T'$, we have to modify (u,v) in and remove (x,y)

Greedy - MST Youtube

☆ greedy alg: repeated makes the locally best choice, ignoring effect on the future

- a cut in graph is a partition of its vertices into 2 non-empty sets
- a crossing edge connects a vertex in 1 set w/ a vertex in another

⇒ given any cut, the crossing edge of min weight is in the MST

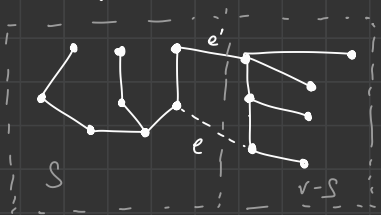


Given a weighted, connected graph $G=(V,E)$. Assume all edge costs are distinct. Let:

S: non-empty proper subset of V

$e=(v,w)$ the min cost edge btwn S and $S-V$

then every MST contains e



→ Proof:

Proof of cut property

Sketch of proof by contradiction:

- Suppose that T is a minimal spanning tree that does not contain e where e is a minimal edge between a set S and V-S. Will show this leads to a contradiction.
- We want to find an e' in T so that exchanging e for e' creates a new spanning tree T' of with less cost than T
  - » There must be a path from v to w in T, creating a cycle in $T \cup \{e\}$
  - » Follow the path to find a different edge that goes from S to S-V
  - » Let that be e', remove it from $T \cup \{e\}$ to get T'
  - » Finally show: T' is a spanning tree and has lower cost

Exchange method

Prim's

Sketch of Proof of Prim's correctness

Proof by contradiction

- Assume that Prim's algorithm does not give an MST
- Let M be a MST for a weighted graph G = (V,E)
- Since Prim constructs a sequence of trees T_i where T_i contains i vertices and i-1 edges. Let T_{k+1} be the first tree that has an edge not contained in the MST.
- But now apply the cut property to T_k and $V - T_k$
  - » The minimum cost edge from T_k to $V - T_k$ must be in the minimum spanning tree
  - » But Prim's algorithm chooses that edge by definition of the algorithm
  - » Thus T_{k+1} must have the same edges as the MST
  - » This contradicts the premise that Prim's algorithm does not find a MST

Greedy ← Kruskal's → sort edges in increasing weight order. Rule: if connect new vertices (doesn't form a cycle), add the edge
Prim's → lightest weight edge that adds a new vertex to current component. Rule: just add.

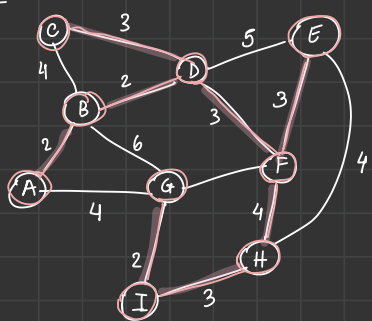
Prim's algorithms

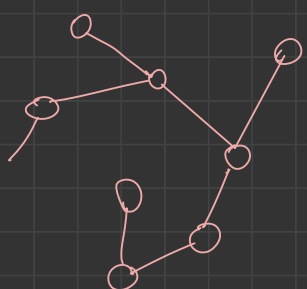
- start from any single vertex a, let $S = \{a\}$; $T = \emptyset$
- find a light edge (u, v) crossing the cut $(S, V-S)$; then $T = T \cup \{(u,v)\}$; $S = S \cup \{v\}$
- if $|T| < n-1$, repeat the step

how to identify light edge ? a new cut evolves from old cut; a little edge crossing the new cut maybe identified w/ little effort.

Prim's ⟶ MST

ex:





T: set to store MST

```
function prim (G, w)
1. for all u in V
2.      cost(u) = infinity;
3.      prev(u) = nil;
4. pick an arbitrary vertex s
5. cost(s) = 0;
6. T = empty_set;
7. H = makequeue(V);        → priority queue of all V
8. while H is not empty
9.      u = dequeue(H);
10.     T = T U {(prev(u), u)};
11.     for every (u, v) in E
12.         if cost(v) > w(u, v)
13.             cost(v) = w(u, v);
14.             prev(v) = u;
15 return (T, prev)
```

Proof that T generated by prim is a MST:

★ proving a more general claim: At every iteration of the while loop, T is contained in some MST.

★ Proof:

- Base case: $k=0$

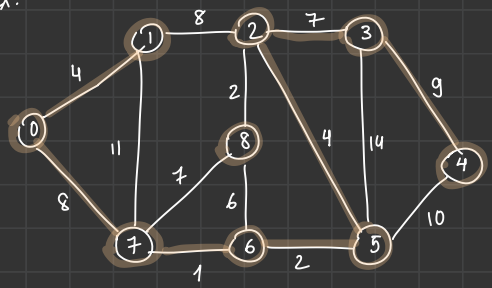
the algo has yet to enter the while loop, then $T = \emptyset$. Therefore, it is contained in every MST

- Assumption: at iteration k, $T \subseteq \mathcal{T}$ for some MST $\mathcal{T}$
- Induction: at iteration $k+1$, $T' = T \cup \{(u,v)\}$, where edge (u,v) is a light edge crossing cut $(S, V-S)$, and S is exactly the set of those vertices in T.
  - ① If $(u,v) \in \mathcal{T}$, then $T' \subseteq \mathcal{T}$, we prove the claim!
  - ② Otherwise, $\mathcal{T}$ has to contain a diff edge (x,y) crossing the cut $(S, V-S)$
- Let $\mathcal{T}' = T \cup \{(u,v)\} - \{(x,y)\}$. $\mathcal{T}'$ is also an MST.

★: TC = $O(|E| + |V| \log |V|)$ ⟹ if use binary heap as priority queue ⇒ $O((|V|+|E|)\log|V|)$

use fibonacci heap as priority queue ⇒ $O(|E| + |V|\log|V|)$

ex:



	✓	✓	✓	✓		✓	✓	✓	✓
vertex	0	1	2	3	4	5	6	7	8
key	0	~~∞~~ 4	~~∞~~ ~~8~~ 4	~~∞~~ ~~14~~ 7	~~∞~~ ~~10~~ 9	~~∞~~ 2	~~∞~~ 1	~~∞~~ 8	~~∞~~ ~~7~~ ~~6~~ 2

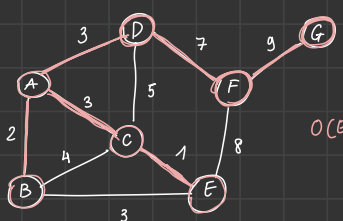
Kruskal's algorithm (TC: ElogE)

```
function Kruskal (G=(V, E), w)
1. Sort edges by weight in the nondecreasing order;
2. forest F = emptyset;
3. for every edge (u, v) in the sorted order;
4.      if u and v not belonging to the same tree in F
5.              F = F U {(u, v)};
6.              update forest F;
```

Steps:

1) Sort edges by ascending edge weight.
2) Walk through the sorted edges and look at the two nodes the edge belongs to, if the nodes are already unified we don't include this edge, otherwise we include it and unify the nodes.
3) The algorithm terminates when every edge has been processed or all the vertices have been unified.

ex:



O(ElogE)

★ Disjoint-set.

- makeset(x): creates a set of single elem x.
- findset(x): identify the set that contains elem x
- union(x, y): union 2 sets containing x and y into one

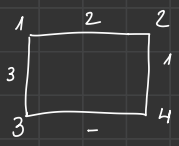
```
function Kruskal (G=(V, E), w)
1. Sort edges by weight in the nondecreasing order;
2. for every u in V,
3. make_set(u);
4. for every edge (u, v) in the sorted order;
5. if find(u) not = find(v)
6. F = F U {(u, v)};
7. union(u, v);
```

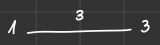
TC of Kruskal's: O(ElogV) or O(ElogE)

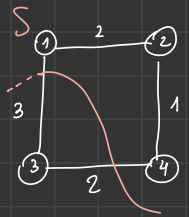
★ Notes between TC of Prim's vs Kruskal's algorithms:

TC of Prim's: $O(V \log V + E \log V) = O((V+E) \log V)$ using binary heap / adjency list
$O(E + V \log V)$ using fibonacci heap

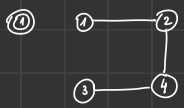
TC of Kruskal's: $O(E \log V)$







Two ideas

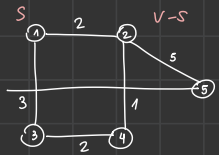


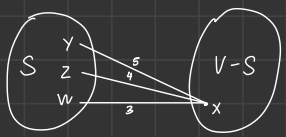
② Soft edges

(2,4), (1,2), (3,4)

(1,3)

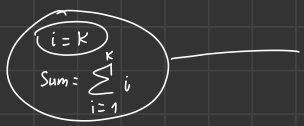
(2,5)





A = B, if

loop-invariant

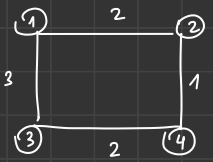


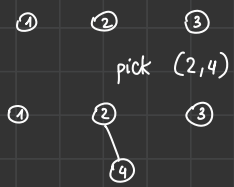
sum = 0

for i = 1 to n

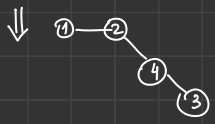
sum = sum + i

$\sum_{i=1}^{n} i$

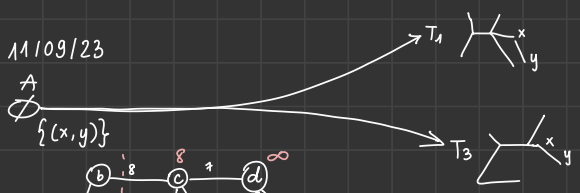


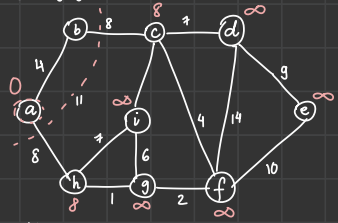


(3,4)

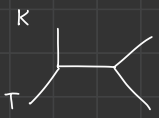


11/09/23



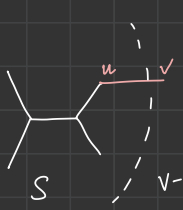


K

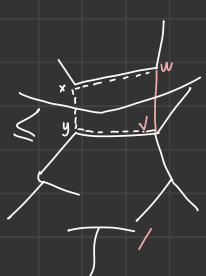


|Le|

$T' = T \cup \{(u,v)\}$



$w(x,y) \geq w(u,v)$



Greedy for fractional knapsack:

- input:
  - n items
  - v_i value
  - s_i size
  - knapsack size W

not only options of items but also options of fractions!

output: $A \subseteq \{1, 2, 3, \dots n\}$ and $0 < f_i \leq 1$ for $i \in A$ st

$\sum_{i \in A} f_i v_i$ is the maximum, subject to $\sum_{i \in A} f_i s_i \leq W$

Greedy: - compute "value density" $d_i = \frac{v_i}{s_i}$

greedy-choice property: the maximum fraction $\min\left\{\frac{X}{s_i}, 1\right\}$ of available item i is included in some optimal solution where X is the space not occupied by items of higher densities.

- Alg:
  1. Calculate the ratio (value/weight) for each item
  2. Sort items in descending way based on ratio
  3. Take the item w/ the highest ratio & add them until we can't add the next item as a whole
  4. At the end add the next item as much (fraction) as we can.

$O(n \log n)$

- nlogn to sort items
- n to traverse and pick from list of items until knapsack is full

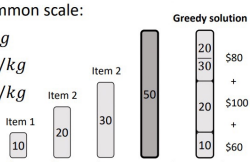
Greedy: keep taking item w/ the highest ratio

Fractional knapsack greedy approach

We will attempt a greedy approach for the fractional problem. What does that mean? We will prioritise items by their value per kg.

We'll call this an item's priority (or profit), p_i. This is a normalised value and with it we can compare items on a common scale:

1. $v_1 = \$60$, $w_1 = 10$ kg, $p_1 = {}^{60}/_{10} = \$6/kg$
2. $v_2 = \$100$, $w_1 = 20$ kg, $p_2 = {}^{100}/_{20} = \$5/kg$
3. $v_3 = \$120$, $w_1 = 30$ kg, $p_3 = {}^{120}/_{30} = \$4/kg$



Fractional knapsack greedy algorithm

```
procedure FracKnap(S, V, W, W_max)          // Items, Values, Weights, max weight
1:   Initialise priority queue Q            // Initialise empty
2:   for each s_i ∈ S
3:       p_i = v_i/w_i                      // pi is value to prioritise
4:       Q.Enqueue(s_i) using pi as priority // Max-priority queue
5:   current_weight = 0                     // Thief's current weight
6:   knapsack = ∅
7:   while current_weight < W_max           // Keep taking while we can carry more
8:       s_k = Q.Dequeue()                  // Get item with max profit
9:       x_k = min(w_k, w_max - current_weight) // How much of item's weight can we take?
10:      current_weight = current_weight + x_k
11:      knapsack = knapsack U {x_k/w_k * s_k} // Add weight fraction of item s_k
end procedure
```

Activity Scheduling - Greedy

★ Input: n activities, each w/ start time s_i and finish time f_i;
Output: max number of activities allowed to use a venue exclusively.

Greedy choice property: the activity w/ the earliest finish time is contained in some optimal scheduling.

★ Algorithm: Starting from the empty schedule, so long as at least 1 candidate exists
↳ always adds in the job / task w/ the earliest finish time

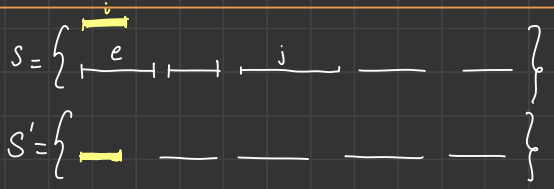
Steps:

① Sort the activities according to their finishing time
② Select the first activity from the sorted array
③ Do the following for remaining activities in the array:
— if the start time of this activity is greater than or equal to the finish time of previously selected activity, then select it.

Pf:

let i be the activity of the earliest finish time
let S be an optimal scheduling $f_i \leq f_j$ for all other j
① if $i \in S$, done!
② if $i \notin S$, let $e \in S$ be the activity of the earliest finish time in S → e: last prev activity in S (optimal scheduling group)

$$f_i \leq f_e \leq s_j \Rightarrow S' \text{ is a valid scheduling}$$
$$\& \ |S'| = |S|$$



- assume activities are sorted in the order of their finish time
  ↑ non-decreasing

function activity-scheduling (n, s, f)
  let $S = \{1\}$; last = 1;
  for i = 2 to n while
    if ($f_{last} \leq s_i$)
    then $S = S \cup \{i\}$
      last = i

11/14/23

NP-Completeness

- Tractable problems: solvable in time $O(n^k)$ for constant k.
  - ★ Reachability: determine if $\exists$ path $s \rightsquigarrow t$ in a given graph.
- Intractable problems: those seemingly w/out polynomial time algorithms.
  - ★ Hamiltanion path: _ Input: $G(V, E)$
    - Output: Yes if $\exists$ (hamiltanion path in G.) ← a path connecting all vertices

search algorithm

Boolean formula satisfiability (SAT): (NP-Complete)

A circuit can be viewed as a tree T, where each leaf is a variable, and each internal node is a logical operation (AND, OR, NOT). The idea is that for any problem in NP, we can by definition take as input an assignment of variables and answer in a polynomial number of operations whether or not this assignment is valid or not.

- Input: a boolean formula $\phi(x_1, \ldots, x_n)$ of n variables
- Output: "Yes" if and only if ϕ is satisfiable.

ex: $f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3)$

★ SAT:
- a decision problem
- no polynomial time algorithms have been found. Even w/ a TC: $O(n^{1000})$ is seemingly impossible
- the first problem was proved NP-complete.
- can be solved using simple exhaustive search.

★ Exhaustive Search:

- Solve SAT using recursive case:

$$f(x_1, \ldots, x_{n-1}, x_n) = f(x_1, \ldots, x_{n-1}, T) \vee f(x_1, \ldots, x_{n-1}, F)$$
$$f(x_1, \ldots, x_{n-1}, F) \Rightarrow g(x_1, \ldots, x_{n-1})$$
$$f(x_1, \ldots, x_{n-1}, F) \Rightarrow h(x_1, \ldots, x_{n-1})$$

Exhaustive TC: $T(n) = O(2^n)$

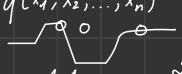
↓

search thru all truth assignment, one by one, but for formulas with n variables

$T(n) = 2T(n-1) + cn$

$\Rightarrow T(2^n)$

⇒ SAT is NP can be verified in poly time.



self-reducible

ex: $f(x_1, x_2, T) = (x_1 \vee x_2 \vee F) \wedge (\neg x_1 \vee T)$

$= (x_1 \vee x_2)$

$(x_1 \vee x_2 \vee \neg x_3)$

T T F T

$= T$

$\vec{x} = (x_1 \ldots x_n)$

$f(\vec{x}, 5)$

$f(x_1, x_2, \ldots, x_5)$

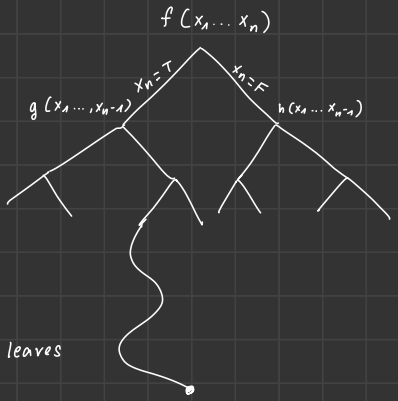
- efficient/polynomial time algs:
  $O(n)$ $O(n \log n)$ $O(n^3 \log^2 n)$
- inefficient algs:
  $O(2^n)$ $O(n!)$ $O(1.000001^n)$

- a decision problem is a problem with a yes/no ans.
- The class NP consists of all decision problems where "yes" answered can be verified efficiently.

Why is SAT ∈ NP Complete:

Proof has 2 steps — Show SAT in NP

Show that every NP problem can be reduced to an instance of a SAT problem in a polynomial time many one reduction.



Optimization problem

In leaves

Max Independent set:

- Input: $G(V, E)$
- Output: $I \subseteq V$, where $\forall u, v \in V, (u, v) \notin E$, st. $|I|$ is maximum.

$T(n) = O(1.619^n)$

Independent Set (decision version)

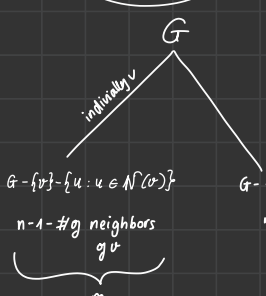
- Input: $G = (V, E)$ and k
- Output: "Yes" if G has independent set of size k.

- Max Independent Set is solvable in $O(n^d)$

$\Rightarrow$ Independent Set is solvable in $O(n^d)$

- Independent set is solveable in $O(n^c)$

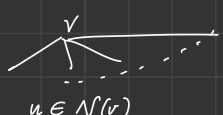
$\Rightarrow$ Max Independent Set is solvable in $O(n^{c+1})$



To find I

① $v \in I$

② $v \notin I$

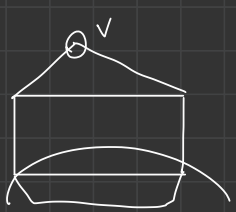


$T(n) = T(n-1) + T(n-1-m)$

$= T(n-1) + T(n-2)$

→ guarantees $m \geq 1$

preprocess G to remove all vertex x $|N(x)| = 0$ and put x in I



$T(n) \leq 1.5^n$

$1.5^{n-1} + 1.5^{n-3}$

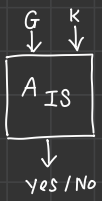
$= 1.5^{n-3}(1.5^2 + 1)$

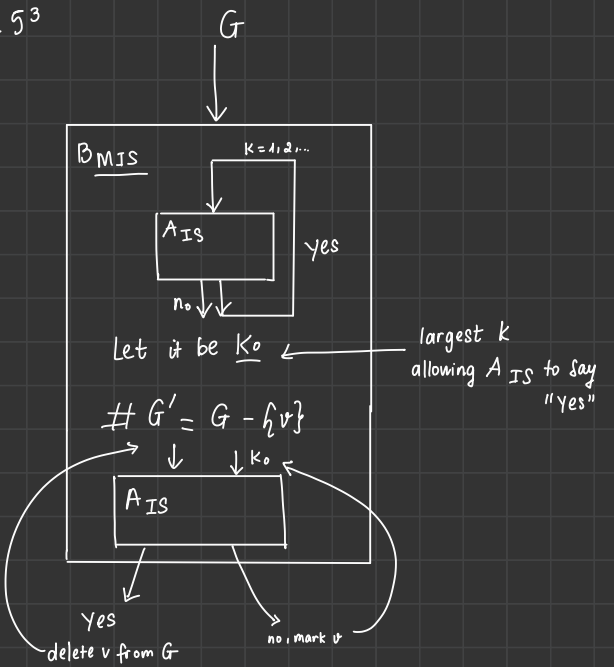
$= 1.5^{n-3}(2.25 + 1)$

$= 1.5^{n-3} \times 3.25$

$\leq 1.5^{n-3} \times 1.5^3$

$= 1.5^n$





★ P: set of problems that can be solved in polynomial time.

★ NP: set of problems that can be verified in polynomial time.

$P \subseteq NP$

★ NP Complete: problems which are in NP but are as hard as any other problem in NP.

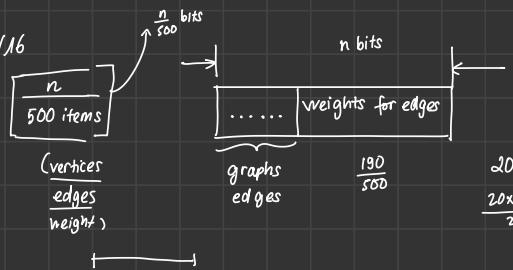
↳ hardest problems in NP.

2. Decision and Search Problems

- Both Hamilton and SAT are decision problems.
- Max Independent set is an optimization problems.

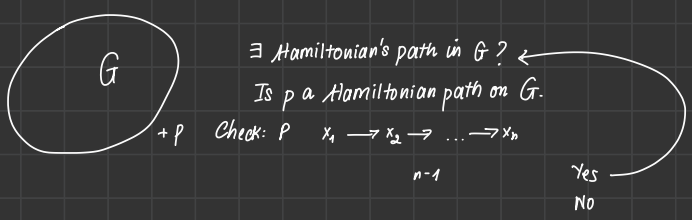
] to investigate tracibility, it suffices to study decision problems

11/16



3. Polynomial solvable v verifiable

★ Hamiltonian cannot be answered in polynomial time, but an answer can be verified in polynomial time.



★ SAT can be verified in polynomial time.

Is $\varphi(x_1 \dots x_n)$ satisfiable ?

\+ A $x_1 \to T$, $x_2 \to F$, ⋮ Does A satisfy φ ?

} usually in polynomial time $O(m+n)$ (# of vars, # clauses)

★ Independent set.

D Decision: Input : G + k ✗

Does G have an independent set of size k? D

✓ Verification

G; k, I, is I an i.s. for G of size k.

x y

Def: NP is the class of decision problems whose answers can be verified in polynomial time.

That is, for every decision problem $D \in NP$, which decides on input x to answer "yes" or "no", there exists a verifier V_D such that

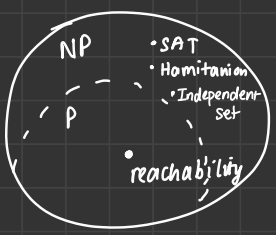
$$\forall x, D(x) = \begin{cases} = \text{"yes"} & \exists y, V_D(x, y) = \text{TRUE} \\ = \text{"no"} & \forall y, V_D(x, y) = \text{FALSE} \end{cases}$$

where V_D can be computed in polynomial time, and y is called certificate / witness to a yes answer!

Note: D may not be computed in polynomial time.

Decision in polynomial time $\xrightarrow[\text{imply}]{\text{can always}}$ verification in polynomial time.

$$D(x) = \begin{cases} \text{Yes} \\ \text{No} \end{cases} \quad ? \qquad V(x, \not{y})$$



$P \subseteq NP$

$\Rightarrow$ Reachability $\in NP$

Knapsack: NP-Hard

Max IS

Input: G

IS:

Input: G, K

Answer: "Yes" iff G has an i.s. of size $\geq$ K

If k=1 : yes
2 : yes
5 : yes
6 : No

Max IS

Input: G

IS:

Input: G, K

Answer: "Yes" iff G has an i.s. of size $\leq$ K

If k=1 : yes
2 : yes
5 : yes
6 : yes

★ Reachability in NP:

- Is SAT

11/16/23

4. NP-completeness theory

$D_1(x)$ ↑

$\Sigma = \{0,1\}$ $\quad \{0,1\}^*$ → Transitive closure

universe

all binary strings

Reduction for decision problems:

Def: let D_1 and D_2 be 2 decision problems. A mapping $f: \Sigma^* \to \Sigma^*$ is a reduction from D_1 to D_2, if for any $x \in \Sigma^*$,

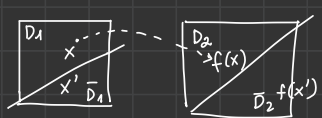
$D_1(x)$ = "yes" iff $D_2(f(x))$ = "yes" where $\Sigma = \{0,1\}$

$\Rightarrow$ denoted as $D_1 \leq D_2$

That is,

- to answer questions D_1 on X
- the transformation converts x to $y = f(x)$ via mapping f
- ans to D_2 on y is then used as answer to x for D_1.

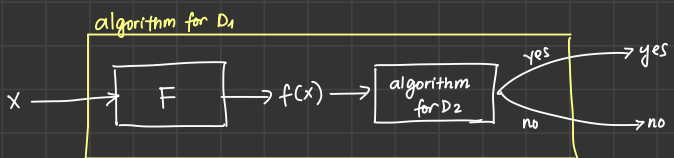
$f: \Sigma^* \longrightarrow \Sigma^*$



$D_1 \subseteq \Sigma^*$

$\bar{D}_1 = \Sigma^* - D_1$

D_1 consists of all positive instances (ie. those w/ "yes" answer)



SA

SAT

$(x_1 \cup x_2)$

$\vdots$

$(x_1 \cup x_2) \wedge (\bar{x}_1 \cup x_2) \wedge (x_1 \cup \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$

Assume:

A_2 for D_2 runs in time $O(n^3)$

F runs in time $O(n^2)$

A_1 for D_1, input x, of length $|x| = n$

time for $A_1 = O(n^2) + O(|f(x)|^3)$ ($|f(x)|$: input size to A_2)

$= O(n^2) + O((n^2)^3)$

$= O(n^6)$

ex

Transformation from SAT and Independent Set (IS)

- need a mapping function $f: \Sigma^* \to \Sigma^*$;
- for any input ϕ, $f(\phi) = \langle G_\phi, k_\phi \rangle$
- satisfying:

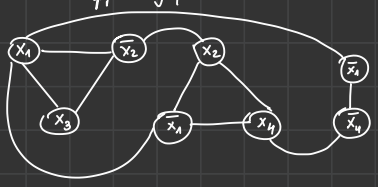
$$SAT(\phi) = \text{"yes"} \iff IS(\langle G_\phi, k_\phi \rangle) = \text{"yes"}$$

- Many decision problems are in NP, but not known to be in P.

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_4)$$

mapped by f to ↓

$G_\phi =$



$k_\phi = 3$

- ϕ is satisfiable $\iff$ G_ϕ has ind. set of size ≥ 3

f follows the rule: (1) map every clause to a complete graph

(2) connect vertices formed by a variable and its negation

11/21/23

positive results

$O(2^n)$ $O(1.5^n)$ $\cdots$ $O(n^{10})$ $O(n^3)$

$O(n^{2.9})$

Negative results

lower bound proof

$\Omega(n)$

Hamiltonian Path:

Input: G — y: evidence of a hamiltonian path

Output: "yes" iff $\exists$ a hamiltonian path in G

NP = Hamiltonian, SAT, IS

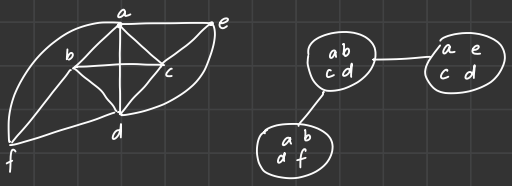
Prove: $P \subseteq NP$

that is to show that for any problem $D \in P$, $D \in NP$

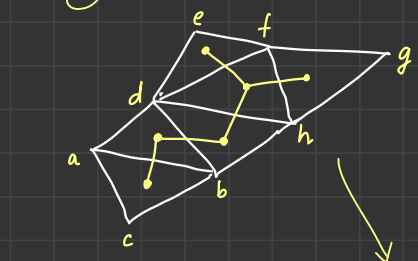
$D(x) = \begin{cases} yes \\ no \end{cases}$ in polynomial time

$A_D(x) = y/n$

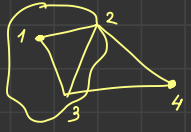
k-tree:

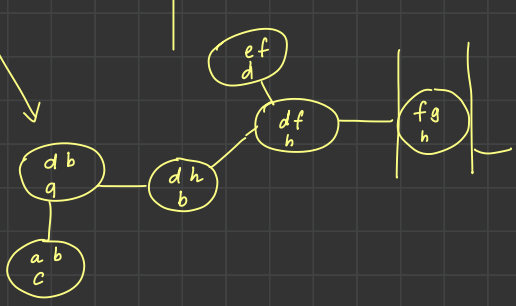


(2)-tree



tree-decompose





tree width

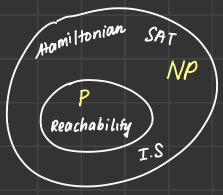
TD-based DP

Max Ind Set

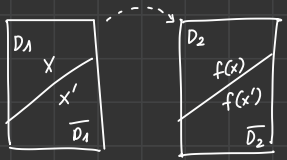
$O(n)$ on trees

1-tree

$O(2^{3+1}n)$



answer to D_1 on $X \Leftrightarrow$ answer to D_2 on $f(x)$



$D_1 \cap D_2 = \emptyset$

$D_1 \mid D_2 = \Sigma^*$

SAT:

Input: φ – boolean formula

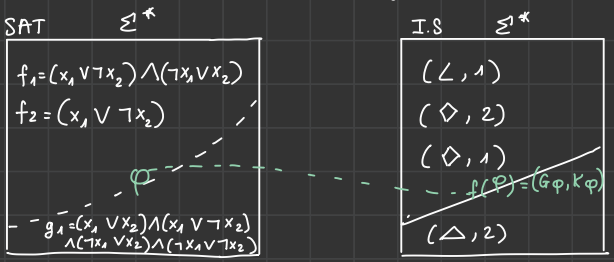
Output: "yes" iff φ is satisfiable

---

in terms of languages:

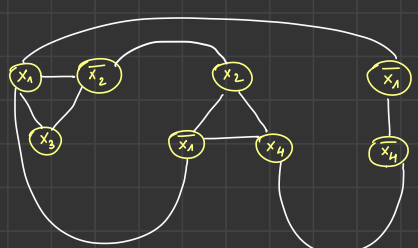
$SAT = \{ \varphi : \varphi \text{ is satisfiable} \}$

$\overline{SAT} = \{ \varphi : \varphi \text{ is not satisfiable} \}$



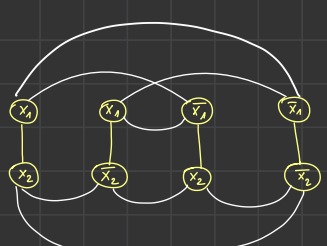
$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_4)$

3-clique



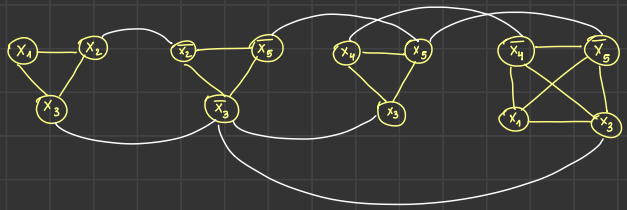
$(G_\varphi, K_\varphi = 3)$

$\varphi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$

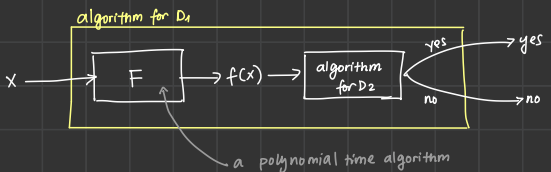


$(G_\varphi, 4)$

$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_5}) \wedge (x_3 \vee x_4 \vee x_6) \wedge (\overline{x_4} \vee x_1 \vee x_3 \vee \overline{x_5})$



Definition: let f be mapping realizing $D_1 \leq D_2$. Reduction is called a **polynomial-time reduction** if $f(x)$ can be computed in time $O(|x|^c)$ for some fixed constant c. It's denoted with $D_1 \leq_p D_2$



Theorem: Let $D_1 \leq_p D_2$. If $D_2 \in P$, then $D_1 \in P$ also.

- Fact 1: $SAT \leq_p IS$
- Conclusion: if IS is in P, so is SAT

Clique:

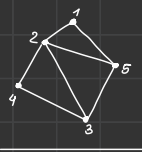
Input: G, K

Output: "yes" iff $\exists$ a clique of size K in G

Let $G = [V, E)$ be a graph

Thus complementary graph of G, denoted with $\overline{G}$ is $\overline{G} = (V, \overline{E})$

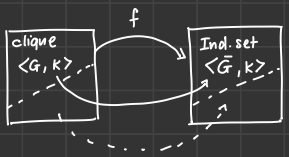
where $\overline{E} = \{(x,y) : (x,y) \notin E\}$



$K = 2$

a clique of m vertices in K_m

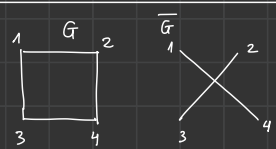
Claim #1: Clique $\leq$ Ind.set



Conclusion: G has a clique of size k iff $\overline{G}$ has an ind. set of size k

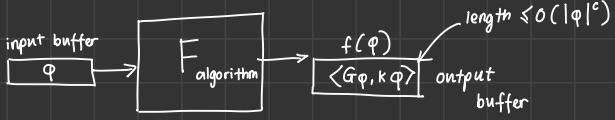
Claim # 2:

Ind. set $\leq$ Clique



Claim SAT $\leq$ Ind. Set

$\varphi \xrightarrow{f} f(\varphi) = \langle G_\varphi, k_\varphi \rangle$



if F runs in time $O(|\varphi|^c)$, $O(n^c)$ $\quad n = |\varphi|$

* Total time for algorithm for D_1

$= O(|x|^3) + O(|f(x)|^2)$ $\quad |f(x)| = O(|x|^3)$

$= O(|x|^3) + O((|x|^3)^2)$

$= O(|x|^3) + O(|x|^6) = O(|x|^6)$

* Therem: Let $D_1 \leq_p D_2$
if $D_2 \in P$, then $D_1 \in P$

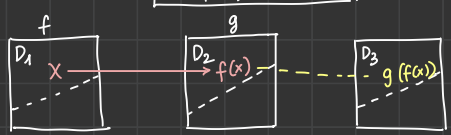
"not easier than"

11/28/23

* Transitivity of $\leq_p$: If $D_1 \leq_p D_2$ and $D_2 \leq_p D_3$ then $D_1 \leq_p D_3$

Proof: $D_1 \leq_p D_2$ $\quad D_2 \leq_p D_3$

$D_1(x) =$ "yes" iff $D_2(f(x)) =$ "yes" and $D_2(f(x)) =$ "yes" iff $D_3(g(f(x))) =$ "yes" $\quad h = f \cdot g$



$h(x) = g(f(x))$

is p-time compntable $\Rightarrow$ $x \to F \to f(x) \to G \to g(f(x))$

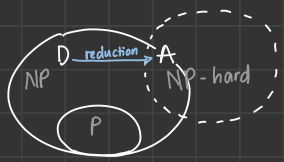
$O(|x|^c) \quad + \quad O(|f(x)|^d)$

$n = |x| \quad = O(|x|^c) \quad + \quad O(|x|^{cd}) = O(n^{cd})$

* Notes: - SAT, IS, Clique, Hamiltonian path and many others can be polynomially reduced to each other.
- all NP

Definition: Decision problem A is called NP-hard if for every problem $D \in NP$, $D \leq_p A$

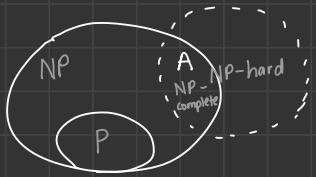
That is, A is "NOT easier than" any problem in NP



Definition: Decision problem A is called NP-complete if

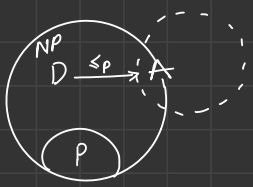
① it is NP-hard

② itself is also in NP



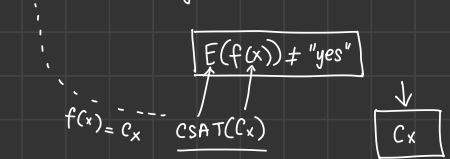
$D_1 \leq_p$

$D_1(x)$ = "yes"



for any x

$D(x)$ = "yes" iff $\exists y \; V_D(x,y)$ = true



$\neg$

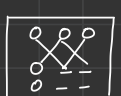
$\vee$

$\wedge$

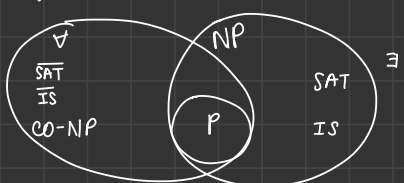
$A \rightarrow B \equiv \neg A \vee B$

$\leftrightarrow$

$$z_1 \leftrightarrow \neg x_1$$
$$\equiv (z_1 \rightarrow \neg x_1) \wedge (\neg x_1 \rightarrow z_1)$$
$$\equiv (\neg z_1 \vee \neg x_1) \wedge (x_1 \vee z_1)$$



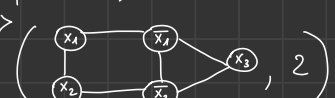
$\forall = E$



$O(NW)$ — knapsack volume

N = # of items

$\varphi = (x_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_1)$



- To prove a problem A to be NP-hard, 2 methods:
  ① directly prove that $\forall D \in NP,\ D \leq_p A$
  ② choose a known NP-hard problem B, and prove $B \leq_p A$.

(ex) **Prove SAT to be NP-hard**

Step: Prove CSAT to be NP-hard first, that is $\forall D \in NP,\ D \leq_p CSAT$.
Then prove $CSAT \leq_p SAT$
$\Rightarrow$ SAT is NP-hard.

★ CSAT: input: boolean circuit C of boolean gates $x_1, \ldots, x_n$;
output: "yes" iff C is satisfiable.

★ NP part of proof:
For every input x, $D(x) =$ "yes" iff $\exists y, V_D(x, y) = TRUE$
iff $\exists y, C(x, y) = TRUE$

Turn $C(x,y)$ to $C_x(y)$ →

For every input x,
$D(x) =$ "yes" iff $\exists y, C_x(y) = TRUE$

$\Rightarrow$ there's a mapping f: $f(x) = C_x$ s.t:
$D(x) =$ "yes" iff $\exists$ assignment y satisfying circuit C_x
iff boolean C_x is satisfiable
$\Rightarrow$ Therefore, we conclude CSAT is NP-hard.

★ Lemma: $CSAT \leq_p SAT$
Proof idea: to transform a circuit C to a boolean formula.
...
(in slides)

Self-Review:

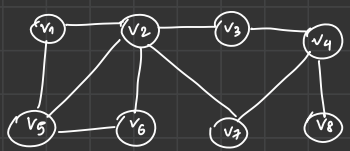
$D_1 \leq_p D_2 \longrightarrow$ D_1 is not harder than D_2

D_2 is not easier than D_1

YOUTUBE

★ IS : Independent Set

An IS is subset V' of the vertices of graph $G=(V,E)$ such that if $u,v \in V'$, then $(u,v) \notin E$.



$\{v_1\}$
$\{v_2, v_4\}$
$\{v_1, v_6, v_3, v_8\}$
} IS

$\{v_1, v_6, v_7, v_4\}$ not IS

★ IS problem: Given a graph G and integer k, does G contain an IS of size k?

Fact 1: IS problem is NP.

Why? We can verify a solution in polynomial time.

Proof: A solution to IS problem is a subset V' of the vertices of G. Let n be the # of vertices of G and m be the # of edges.

Given V' we first verify it contains k elements.

For each pair $u,v \in V'$, check that $(u,v) \notin E$. There are $O(n^2)$ pairs of vertices in V' and m edges, so this algo takes $O(mn^2)$ time - poly in both m and n.

★ Proof: IS is NP-complete:

We can prove IS is NP-complete if we can reduce any NP-complete problem to IS problem.

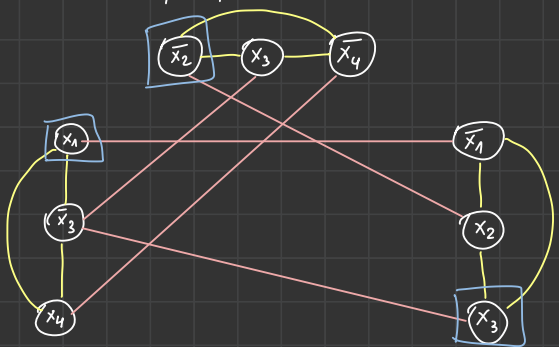
- Goal: 3SAT $\longrightarrow$ IS
- Proof: Let φ be a boolean expression in 3-CNF. We want to express φ as a graph so that if the graph has an ind. set, then φ is satisfiable

$$\varphi = (x_1 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee x_3)$$

① create vertex for each literal

② connect each literal to other 2 literals in the same clause

③ connect each literal x_i to $\overline{x_i}$.



★ 3SAT: Given a boolean expression in 3-CNF, can it be satisfied?

x_i and $\overline{x_i}$ cannot be chosen b/c conflict graph construction

Notes: 3SAT
- find a way to assign 0/1 to the vars s.t the formula evaluate to true. (each clause = True)
- fail if 2 of the literals are in conflict aka pick x_i and $\neg x_i$

☆ The Reduction:

- G_φ will have 1 vertex for each literal in a clause
- Connect 3 literals in clause to form △. The IS will pick at most 1 vertex from each clause, which will correspond to the literal to be set to true.

ex: $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

⇒ φ is satisfiable iff G_φ has independent set of size k (= num of clauses in φ)

☆ Proving a problem is NP-complete

To prove a problem is NP-complete, have to show that it is both in NP and it is NP-hard.

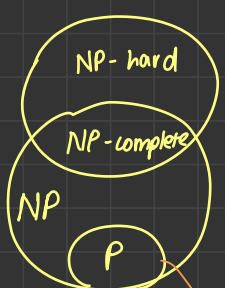
Steps: 1) Show that X is in NP (there's a polynomial time verifier for X)

2) Pick a known NP-complete problem
   - ↳ state problem Y you are reducing to X.
   - ↳ Show $Y \leq_p X$

3) Construct an alg to solve Y given an algo to show X.

4) Prove and conclude polytime.

⇒ Since Y is NP-complete, X is NP-hard, X is in NP, X is in NP → X is NP-complete.



An example shown a lot before: Prove IS is NP-complete.

Revisit:
- First show IS in NP (since given any set S we can check in polytime that S is indepenent that $|S| = k$)
- Show IS is NP-hard via reduction from 3-SAT to IS.
  - ↳ constructs a graph based on 3-SAT
  - ↳ Show that the existence of an independent set in this graph corresponds to a satisfying assignment for 3SAT

every NP-complete problem is NP-hard
NOT every NP-hard problem is NP-comple
(ex: Halting)

Prove a problem X to be NP-hard: 2 ways — Reduce NP-hard problem to X (like the theorem 3)
Reduce any NP-complete prob to X

CSAT and SAT: NP Hard and NP-complete — b/c SAT, CSAT can be proven NP-hard and are also in NP
IS : NP Hard and NP-complete — b/c IS can be proven NP-hard via 3SAT reduction and IS is also NP ⇒ also NP-complete

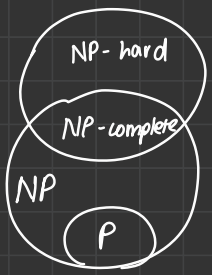
★ Def of complexity classes

★ Problems ∈ P:
- MST
- SPT
- Union find
- Sorting algs
- Fib

★ Problems ∈ NP:
- SAT
- Hamiltonian path
- Clique
- IS
- Travelling sailman

★ Problem Q ∈ NP-complete if: 1. Q ∈ NP
2. Every Q' problem in NP can be reduced to Q in poly time ★

→ SAT
Clique
Ham path



★ NP-complete probs are hard b/c all problems in NP must be reducible NP-complete
(1 NP-complete prob can solve all NP problems)

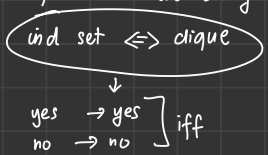
Clique Problem: A clique is a subset V' of the vertices of graph $G=(V,E)$ s.t. if $u,v \in V'$ (for each pair), then $(u,v) \in E$

★ Clique is in NP:

- Takes polynomial time to check answer!

★ IS problem reduces to Clique problem: (in poly time)

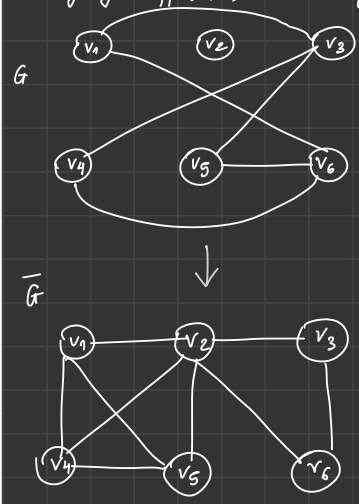
Lemma: if V' is an ind. set of G, then V' is a clique of $\bar{G}$



Proof of reduction:

- Let G be a graph and k be an instance of ind. set problem. Let $\bar{G}$ be complement of G.
- By lemma 1, G has ind. set of size k iff $\bar{G}$ has a clique of size k.
- mapping (G, k) to $(\bar{G}, k)$ is the reduction from the IS to clique problem and computing $\bar{G}$ takes poly-time :)

★ Graph complement: of a graph $G=(V,E)$ is graph $\bar{G}$ with the same vertices as G s.t. (u,v) is an edge of $\bar{G}$ iff (u,v) is not an edge of G

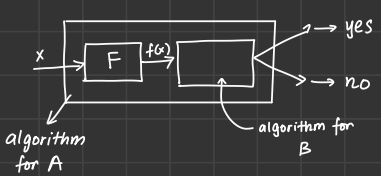


11/30/23

Review Lecture

$A \leq_p B$ {
① $f: \Sigma^{1*} \to \Sigma^{1*}$
② f can be computed in $O(n^c)$ $n = |x|$
③ $A(x) =$ "yes" iff $B(f(x)) =$ "yes"
}

Theorem 1: if $B \in P$, so is A



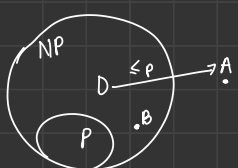
Theorem 2: if $A \leq_p B$, $B \leq_p C$ Then $A \leq_p C$ } transitivity

Theorem 3: if A is NP-hard, $A \leq_p B$ then B is NP-hard

if A is NP-hard
$\equiv \forall D \in NP, D \leq_p A \leq_p B$

Theorem 4: let $A \leq_p B$ if A is NP-complete, then B is NP-hard

★ Let A be an NP-hard problem:
if $A \in P$
then $P = NP$



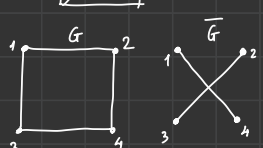
★ Known NP complete problems:
SAT, IS, Clique, Hamiltonian Path, CSAT → NP complete

★ NP-Hardness proofs
① $\forall D \in NP, D \leq_p CSAT$
② $CSAT \leq_p SAT$
③ $SAT \leq_p IS$
④ $IS \leq_p Clique$

★ Clique Problem
Input: G, k
Output: "yes" iff
$\exists$ a complete subgraph of size k in G.

ex)

, 4 ? → "No"



Quiz 9 Scope:

- decision vs verification
- reduction $\leq$, polynomial-time reduction $\leq_p$, implications of $\leq_p$;
- def of NP-hardness, NP-completeness implications of NP-hardness, NP-completeness
- known NP-complete, NP-hard problems and why they are.

11/30/23

Review Lecture

Quiz 5 Review:

① SCC: DFS, G^T, DFS (start from largest post time) } 3 steps

② Single-source shortest paths
- DAG (topological sort ← DFS — reversed order of post)
- Dijkstra's (queue, relaxation) (negative weights? no)
- Bellman-Ford (n-1 rounds of relaxation) (allows negative edges)

Quiz 6:

- Prim's: (queue, specific cut separates those in the queue from those not)
- Kruskal's (sorted edge weights)
  (check for cycles)
  (Disjoint sets)
- Greedy: choice property
  * MST
  * activity scheduling
  } no fractional knapsack

- DP → 4 steps
  - overlapping subproblems
  - optimal substructure (numerical object function)
  - look-up table building
  - tracing back optimal solution

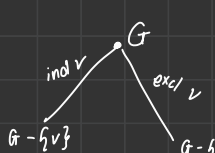
  * fastest assembly path
  * casino decoding
  * coin change
  * 0-1 knapsack
    ↳ edit distance [LCS, pairwise alignment]

Quiz 8:

- intractable problems (tractable $O(n^c)$)

SAT Hamiltonian path Max I.S

- exhaustive search



$T(n) = T(n-2) + T(n-1) + O(n)$ | $n \geq 1$

$T(n-2)$: from including v; $T(n-1)$: from excluding v

- Relationships between decision problems and optimization problems.

-Need to review more.

Quiz 9: Solutions

① Based on the definition of class NP, for any problem $D \in NP$, there is a polynomial-time verifier (algorithm) V_D such that for every input x,

$D(x) =$ "yes" if and only if $\exists y, V_{D(x,y)} =$ "yes"

1) y is called certificate / witness
2) If for some y, $V_{D(x,y)} =$ "no", then answer to $D(x)$ is unknown.
3) If for some y, $V_{D(x,y)} =$ "yes", then $D(x) =$ "yes"
4) Does Hamiltonian path have a polynomial time verifier? yes (Hamiltonian path $\in NP$)
5) Does reachability have a polynomial time verifier? yes (Reachability is P → of course can be verified in P)

② To prove that a polynomial time reduction $A \leq_p B$ exists between problems A and B, we need to prove a few things:

1) There's a mapping $f: \Sigma^* \to \Sigma^*$ between A & B.
2) For any x, $A(x) =$ "yes" if and only if $B(f(x)) =$ "yes"
3) transformation function f can be computed in poly time.

③ Let $A \leq_p B$ be a polynomial time reduction between problem A and B. True / False?

1) If A is NP-hard, B is also NP-hard? True
2) If A is NP-complete, then B is also NP-complete? False we don't know
3) If B is NP-comple, then A is NP-hard. False
4) If B is NP complete, then $A \in NP$ True NP problem A reduces to B.
5) If $B \in P$, then $P = NP$ False ?
6) If $B \in P$, then $A \in P$. True
7) If $A \leq_p SAT$, then $B \leq_p SAT$ False not transitivity !!!
8) If $SAT \leq_p A$, then B is NP-hard True
   ↳ SAT is NP-complete (alr known)
   $SAT \leq_p A$ and $A \leq_p B \Rightarrow \underbrace{SAT \leq_p B}_{\text{NP-complete}} \Rightarrow B$ is NP-hard.

④ 2 methods to prove that a problem is NP-hard:

Let the problem be X:

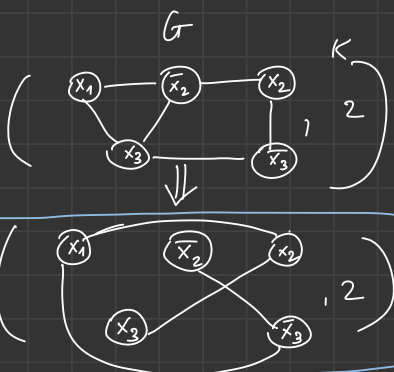
1) To prove $\forall D \in NP$, $D \leq_p A$.
2) To pick an NP-hard problem B and prove $B \leq_p A$.

⑤ $SAT \leq_p$ Clique

* Why? $SAT \leq_p IS$ and $IS \leq_p$ Clique
  $\Rightarrow SAT \leq_p$ Clique (transitivity)

* What will the formula be transformed to by reduction $SAT \leq_p$ Clique.

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3)$$



Quiz 8 Solutions:

④ NP: class of decision problems whose answers can be verified in poly time. That is, for every decision problem $D \in NP$, which decides on input x to answer "yes"/"no", $\exists$ a V_D such that:

$$\forall x,\ D(x) = \begin{cases} = \text{"yes"} & \exists y,\ V_D(x,y) = TRUE \\ = \text{"no"} & \forall y,\ V_D(x,y) = FALSE \end{cases}$$

Let IS be decision problem $D \in NP$.

1) x is an input graph G and a number of vertices in IS
or x is (G, k), where G is a graph & $k > 0$ is an int.

2) y is an independent set of size k in G.

3) V_D: a verification algorithm to check if y is an IS of size k in G.

4) V_D runs in polynomial time $O(N^c)$ for some constants c, where $N = |(x, y)|$.

⑤ Does D run in polynomial time?

→ We are not sure D runs in polynomial time b/c D is verifiable in polynomial time but it doesn't mean P is solvable in polynomial time.

③ Assume V_{SAT} to be a verification alg for SAT.

1) Input of SAT: (ϕ, A): a boolean formula ϕ and truth value assignment to all variables in ϕ.

2) What should V_{SAT} check to fulfill its duty?

Overall: check if given a boolean formula, assign all T/F values from A to the corresponding variables to check if the formula results in a T value.

Detailed steps:

1) verify that A has assignments for all variables in ϕ.
2) Check that ϕ can be evaluated to TRUE
3) output "yes" iff 1) + 2) are true.

② Let MST-D be a decision version of MST.

Now any algo A for MST-D can be used to construct algorithm A_{mst} for problem MST

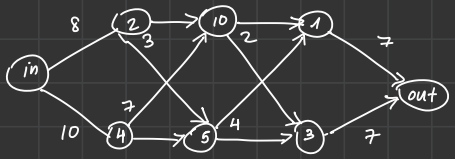
1) On input graph H of MST, how does A_{mst} use A to find the minimum weight w_0 of a spanning tree for H?

→ A_{mst} runs on H and iterate thru all w until answer changed from "no" → "yes"
↓
find the w_0 of a.

2) Once w_0 is found, how does A_{mst} use A again to find the corresponding spanning tree of weight w_0?

3) How to guarantee that, if A has TC $O(n^d)$, the constructed A_{mst} has TC $O(n^{d+2})$?

Quiz 6 Solutions:

① 

line	1	2	3	out
1	10	20	21	28
2	14	18	21	

station #

⇒ both ways work!

Fastest factory path: in → 1 → 2 → 4 → 5 → 3 → out

② Knapsack. Fill table up to k=2

item	1	2	3
size	1	3	2
value	2	4	5

X: capacity / k	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	2	2	2	2	2
2	0	2	2	4	6	6
3			...			

Assume n items, knapsack size W.

1) The traceback to start finding the selected items: (n, W) → the bottom right cell. It ends (0, X) for some X or (k, 0) for some k.

2) If prev points to up cell, it is abandoned. Otherwise, if point up-left, selected.

3) TC of knapsack: $O(nW)$

4) While TC looks in a polynomial time, it may not be a polynomial time in input size. This is b/c W can be as large as exponential of input size.

③ Coin change: objective fun c(m), base c(0)=0

$$c(m) = \min \begin{cases} c(m-25)+1 & m \geq 25 \\ c(m-10)+1 & m \geq 10 \\ c(m-5)+1 & m \geq 5 \\ c(m-1)+1 & m \geq 10 \end{cases}$$

1) 2 cases of c that share an overlapping sub problem: c(11) and c(2) → share c(1)

2) Where in objective func formulation shows that the problem has optimal substrure:

c(m) is optimized over 4 optimal solutions
w/ the min func

3) Assume table cells prev[k] are used to remember the corresponding type of coin chosen in computing c(k).

★ When track back process encounters prev[14] that has info dime (10 cents), it goes to prev[4] next.

★ How was dime put in cell [14]?

④ Edit dis:

$$E(i,j) = \min \begin{cases} E(i-1,j-1) + 0 & \text{when } x_i = y_i \\ E(i-1,j-1) + 2 & \text{when } x_i \neq y \\ E(i,j-1) + 1 \\ E(i-1,j) + 1 \end{cases}$$

1) Dimension of matrix table Ed : $(n+1) \times (m+1)$

2) cell $Ed[4,5]$ represents the edit distance between $x_1 \dots x_4$ and $y_1 \dots y_5$

3)

$$Ed[4,5] = \min \begin{cases} E[3,4] + 2 & \text{when } x_4 \neq x_5 \\ E[3,4] & \text{when } x_4 = x_5 \\ E[4,4] + 1 \\ E[3,5] + 1 \end{cases}$$

4) $Ed[4,5] = 20$, $Ed[3,4] = 20$. $\Rightarrow x_4 = x_5$

5) $Ed[4,5] = 22$, $Ed[3,4] = 20$ $\Rightarrow x_4 \neq x_5$

6) $Ed[3,4] = 22$, $Ed[4,5] = 21$, $Ed[4,4] = 21$

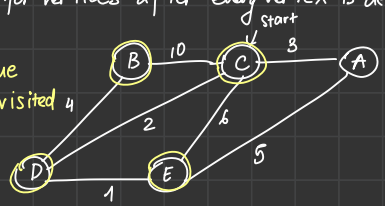
	0	1	2	3	4
	"	a	b	c	d	e
0	"					
1	f					
2	g					
3	e				22	
4	c				21	21

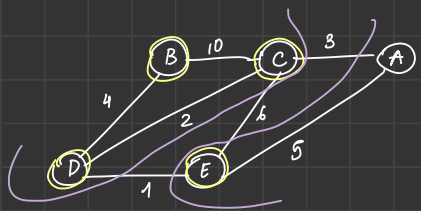
Quiz 7 Solutions:

⑦ Run Prim's algorithm. Show in table change of cost values for vertices after every vertex is dequed.

Vertex	A	B	C	D	E	vertex dequeued
cost	∞	∞	0	∞	∞	
cost	3	10	0	2	6	C
cost	3	4	0	2	1	D
cost	3	4	0	2	1	E
cost	3	4	0	2	1	A
cost	3	4	0	2	1	B

→ dequeue out of unvisited nodes





1)

2)

② Prim's doesn't work well on disconnected non-directed graphs. However, can be fixed.

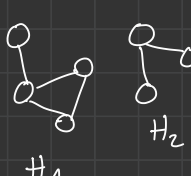
1) Will algo output correctly an mst for component H_1? "yes"

2) After algo is done w/ vertices in H_1, priority queue still has all vertices in H_2 and ∞s

3) vertices with cost ∞ will be dequeued by cost and prev won't be updated.

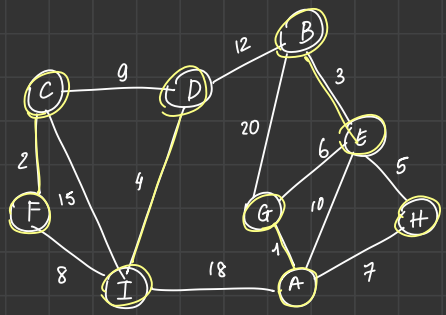
4) when there is vertex w/ cost ∞ to be dequeued, its cost should be set = 0.

Draft:



③ Kruskal's

Edge selected	Disjoint Sets
(A, G)	[A, G] [B] [C] [D] [E] [F] [H] [I]
(C, F)	[A, G] [G, F] [B] [D] [E] [H] [I]
(B, E)	[A, G] [C, F] [B, E] [D] [H] [I]
(D, I)	[A, G] [C, F] [B, E] [D, I] [H]
(E, H)	[A, G] [C, F] [B, E, H] [D, I]
(E, G)	[A, G, B, E, H] [C, F] [D, I]
(F, I)	[A, G, B, E, H] [C, F, D, I]
(D, B)	[A, G, B, E, H, C, F, D, I]



④ Greedy choice property for the activity scheduling problem states that activity a with the earliest finish time belongs to some optimal solution. If a given solution S doesn't contain a, show that the greedy choice property still holds for a:

Try to prove that if a given optimal solution S doesn't contain a, it means that ∃a solution x in S s.t $f_s \geq f_a$. A solution S' can be created by S' = x + a. ⇒ S' has an activity that finishes earlier than S. → S' is more optimal.

* Correct ans:

Let $S = \{e, x_1, \ldots, x_m\}$ where e is the act. w/ earliest finish time in S.

$S' = S \cup \{a\} - \{e\}$.

$x_1, \ldots, x_m$ do not conflict with a → S' is an optimal solution that contains a.