**Week 1**

**Intervals**

Closed: [a,b] = { x | a ≤ x ≤ b }

Open: (a,b) = { x | a < x < b }

Half-open: [a,b) = { x | a ≤ x < b } OR (a,b] = { x | a < x ≤ b }

**Binomial Coefficients (Permutations)**

**Graphs**

Simple – no loops or multiple edges

Hamiltonian cycle – each vertex visited exactly once

Euler cycle – each vertex and edge visited exactly once

Week 4

Abstract data types

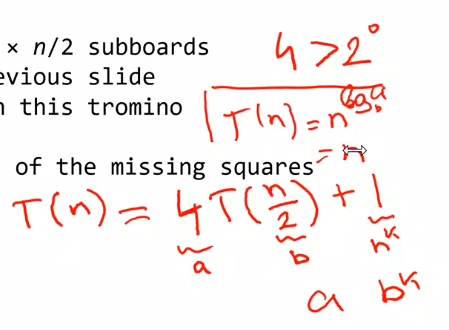
Stacks

* stack\_init(): initialise
* empty(): if stack is empty, true. Else, false.
* push(val): add *val* to the stack.
* pop(): return item on top of the stack and remove it from the stack.
* top(): return item on top of the stack.

Week 5

Every recursive algorithm can be written as iterative.

Week 6

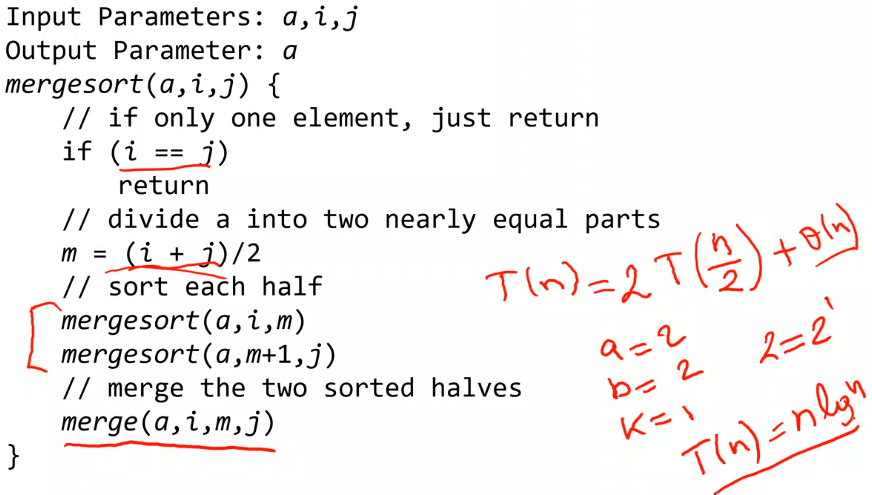


Sorting algorithms

Stable = algorithm keeps original order for equal values.

Lower bound for comparison based sorting is *n\*log n*.

Mergesort recurrence relation



* Each mergesort has 2 recursive calls to mergesort: 2T
* These 2 calls operate on 2 halves of the array: n/2
* complexity of merge is: Ѳ(n)
  + therefore T(n) = 2T(n/2) + Ѳ(n)
* With Master Theorem T(n) = aT(n/b) + f(n)
  + a = 2, b = 2, k = 1
  + a = bk | 2 = 21, therefore Ѳ(n log n)

All comp based

worst = Ω(n lg n)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sort | Best | Average | Worst | Type | Lower | Stable | Comp |
| Merge | T(n log n) | T(n log n) |  | D&C | O(n\*log n) | Y | Y |
| Bubble | O(n) |  | T(n2) | Brute | O(n2) | Y | Y |
| Insertion | O(n) | O(n2) | T(n2) | Dec&C | O(n\*log n) | Y | Y |
| Quick | T(n log n) | T(n log n) | T(n2) | D&C | O(n\*log n), O(n2) (worst) | N | Y |
| Heap | O(n lg n) Ω(n) | T(n log n) Ω(n) |  | Tran&C | O(n\*log n) | N | Y |
| Radix |  | Ѳ(n lg m) | Ѳ(n lg m) |  |  |  |  |
| Counting |  | T(n+m) |  |  |  | Y | N |

Lecture 10

DFS – stack, shortest paths

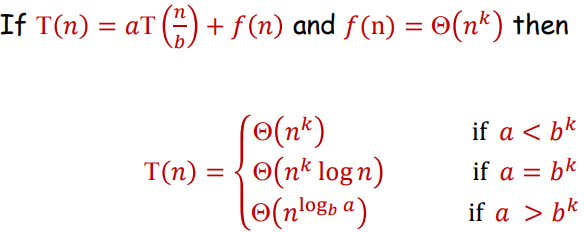
BFS – queue

DYNAMIC PROGRAMMING

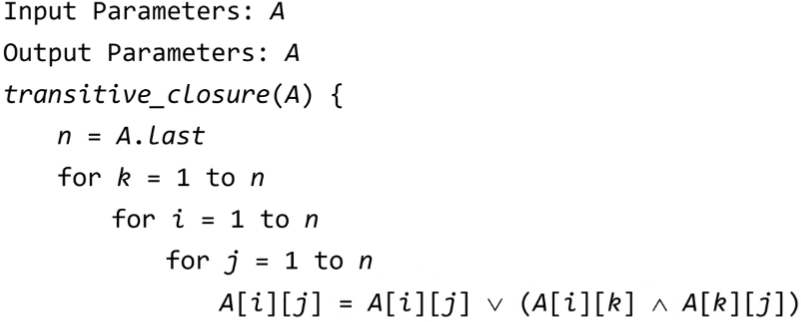
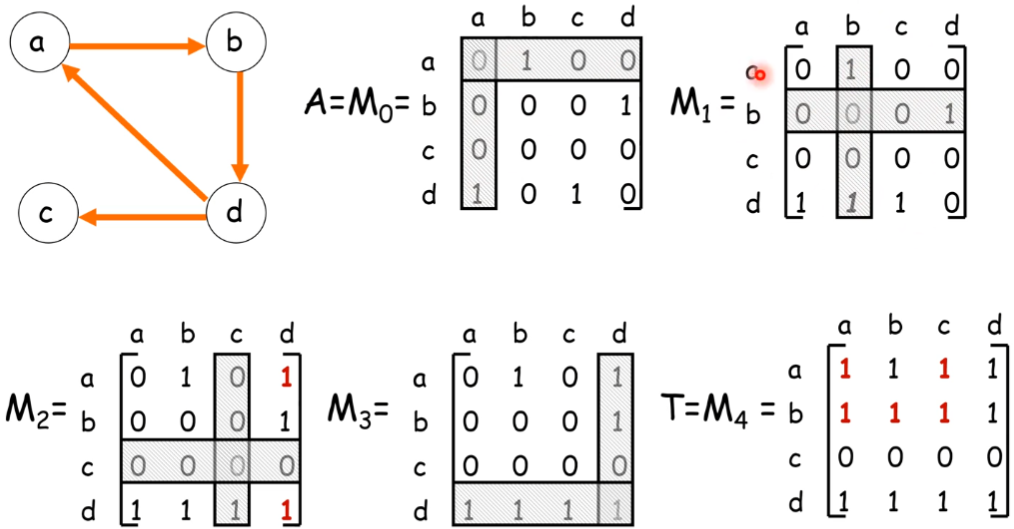
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 0 | y | n | n | n | n | n | n | n | n | n | n | n | n | n | n |
| 1 | y | y | n | n |  |  |  |  |  |  |  |  |  |  |  |
| 3 | y |  |  | y |  |  |  |  |  |  |  |  |  |  |  |
| 13 | y |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | y |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 | y |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

SORTING ALGORITHMS

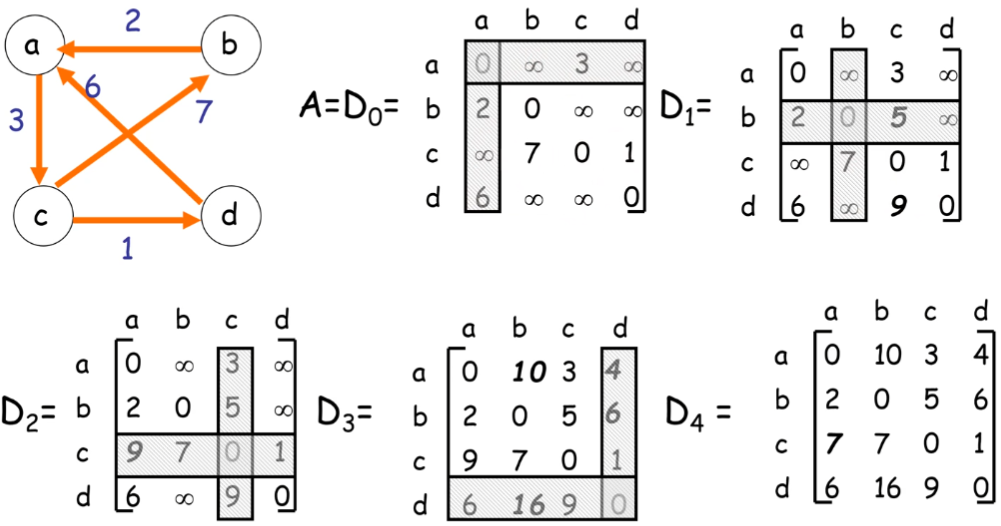
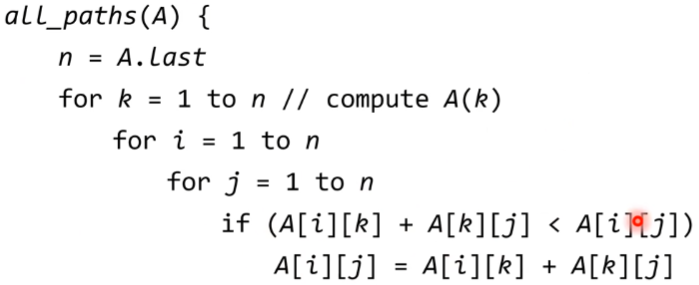
MASTER THEOREM



Warshall’s algorithm



Floyd’s algorithm

Week 2

* 1. increasing and nondecreasing
  2. nondecreasing
  3. none
  4. increasing, decreasing, nonincreasing and nondecreasing
  5. decreasing and nonincreasing
  6. nondecreasing and nonincreasing
  7. Neither
  8. CNF
  9. DNF
  10. CNF & DNF
  11. CNF & DNF

|  |  |  |  |
| --- | --- | --- | --- |
| p | q | Γ(p ^ q) | Γp v Γq |
| T | T | F | F |
| T | F | T | T |
| F | T | T | T |
| F | F | T | T |

|  |  |  |  |
| --- | --- | --- | --- |
| p | q | Γ(p v q) | Γp ^ Γq |
| T | T | F | F |
| T | F | F | F |
| F | T | F | F |
| F | F | T | T |

* 1. 3/2
  2. 5x
  3. 8
  4. 3x

Week 6

1. 1,2,3,4,8,7,6,5,9,10,13,14,11,12,16,15
2. 1,2,5,6,3,7,9,10,15,4,8,11,13,14,16,12
3. 4,10,9,6,3,1,2,8,5,7
4. 1 … n, n = 6

for k to n

used[k] = false

reperm(1,n)

reperm(k,n) {

for s=1 to n

if !used[s]

}

n = 10

Minheap/maxheap  
Week 12

1. m = 3, n = 15, q = 5, r = 4 (22 mod 5)

f[0] = 0

pfinger = 0

for j=0 to 2 {

|  |  |  |  |
| --- | --- | --- | --- |
| j | 0 | 1 | 2 |
| f[0] | 0 | 1 | 3 |
| pfinger | 1 | 2 | 0 |

f[0] = 2\*f[0] + t[j] mod 5

pfinger = (2\*pfinger + p[j] mod 5

}

i=0

while (i+m≤n)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| f[i+1] | 1 | 0 | = | = | = | = | = | = | = |
|  |  |  |  |  |  |  |  |  |  |

I = 2

Upper lower theta notation

𝑓(𝑛) = 𝑂(𝑔(𝑛)) || 𝐶1 > 0 and 𝑁1 such that 𝑓(𝑛) ≤ 𝐶1𝑔(𝑛) for all 𝑛 ≥ 𝑁1

𝑓(𝑛) = Ω(𝑔(𝑛)) || 𝐶1 < 0 and 𝑁1 such that 𝑓(𝑛) ≥ 𝐶1𝑔(𝑛) for all 𝑛 ≤ 𝑁1

a = 2, b = 2, k = 1



 = false



 = false











 = Ѳ(n2)





 = Ѳ(2n)

= Ѳ(n!)

 = not DNF or CNF

2+4+6+…+2n = 2(1+2+3+…n) = 2\*(n(n+1)/2) = n(n+1) = n2 + n = Ѳ(n2)

(1+2+3+…+n) = (n\*(n+1))/2

log ab = b log a

P/N/NP

P = decidable in polynomial time on a deterministic single-tape Turing machine, realistically solvable on a computer **decided in polynomial time**

NP = nondeterministic polynomial 1st stage **verified in polynomial time**

Any problem in P can be reduced in polynomial time to any NP-complete problem

NP only if it is decided by some nondeterministic polynomial time Turing machine.

NP-complete problems = either all in P or none in P, if a polynomial time algorithm exists for any of these problems, all problems in NP would have a polynomial time algorithm

The intersection on P and NP-complete is empty

If A is polynomial time reducible to B (denoted A <=p B) then:

* if A є P, we **don’t know** anything about B / if it is also in P.
* if B є P, A є P
* if A є NP-complete, B є NP-complete.
* if B є NP-complete, we **don’t know** anything about A / if it is also NP-complete.

If there is an NP-complete problem that has a polynomial time algorithm, then P=NP.

If there is an NP-complete problem that has a polynomial time constant factor approximation algorithm, then P != NP.

There is NOT a simple factor 2 approximation algorithm that solves the TSP problem in polynomial time.

Proving NP-complete

Prove NP first (gettable with some algorithm)

P =/= NP

It is not known whether P is equal to/proper subset of NP

Any problem in NP can be reduced in polynomial time to any NP-complete problem (but not any problem in P).

The intersection on P and NP-complete is not empty.

If a polynomial time algorithm exists for any problem in NP, then all NP-complete problems would NOT have a polynomial time algorithm.

NP is the class of decision problems that are decidable in polynomial time on a non-deterministic Turing machine – have polynomial time verifiers.

3SAT

Turing Machines

Time/space complexity = amount of time/space required to complete, if time com is n, space com is no more than n + 1

The complexity of a problem is in class Ѳ(f) if it can be solved by algorithms of class Ѳ(f) and any better solution has class Ѳ(f).

If we restrict instances of the TSP to Euclidean instances, then there exist a constant factor approximation algorithm for solving this problem.