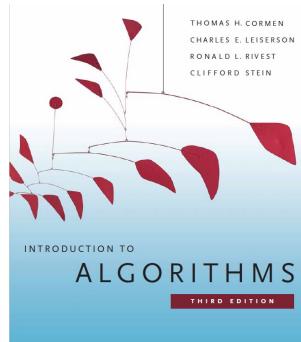




# Randomization and Quicksort

Prepared by Shmuel Wimer  
Courtesy of Prof. Dror Rawitz



Aug 2024

Algorithms and DS II: Randomization and Quicksort

1

1



# Worst Case Run Time

Given **deterministic** algorithm  $A$ , let  $t_A(x)$  be its run time on input  $x$ .

The performance of  $A$  w.r.t the input size  $n$ , denoted  $T_A(n)$  is

$$T_A(n) = \max_{\{x:|x|=n\}} t_A(x)$$

Let  $f: \mathbb{N} \rightarrow \mathbb{N}$ . The run time complexity of  $A$  is  $O(f(n))$  if  $\exists n_0 \in \mathbb{N}$  and  $\exists C > 0$  s.t.  $\forall n \geq n_0$  there is

$$T_A(n) \leq C \cdot f(n)$$

Aug 2024

Algorithms and DS II: Randomization and Quicksort

2

2

1



Let  $D_n$  be the **distribution** of inputs of size  $n$ . It is of interest to bound the run time expectation.

$$T_A(n) = \mathbb{E}_{x \sim D_n}[t_A(x)]$$

$D_n$  knowledge is required. Uniform distribution is usually assumed.

Given **probabilistic** algorithm  $A$ , let  $t_A(x)$  be its run time on input  $x$ .

$t_A(x)$  is random variable, depending on  $A$ 's probabilistic decisions. Hence we consider  $\mathbb{E}[t_A(x)]$ , and worst case run time of  $A$  is

$$T_A(n) = \max_{\{x:|x|=n\}} \mathbb{E}[t_A(x)]$$



## Quicksort (C.A.R. Hoare 1962)

Divide-and-conquer algorithm comprising three steps. Given  $A[p..r]$ ,

- **Divide:** Partition  $A[p..r]$  into  $A[p..q - 1]$ ,  $A[q]$ ,  $A[q + 1..r]$  s.t.  $A[p..q - 1] \leq A[q] < A[q + 1..r]$ .
- **Conquer:** Sort  $A[p..q - 1]$  and  $A[q + 1..r]$  recursively.
- **Combine:** No work required since  $A[p..r]$  is already sorted.

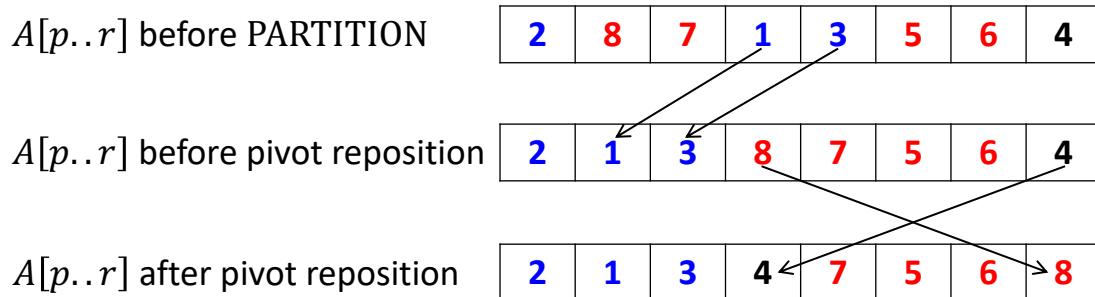
```

QUICKSORT( $A, p, r$ )
  if  $p < r$ 
     $q = \text{PARTITION}(A, p, r)$ 
    QUICKSORT( $A, p, q - 1$ )
    QUICKSORT( $A, q + 1, r$ )
  
```



PARTITION procedure rearranging  $A[p..r]$  is crucial, specifically the **pivot**  $q \in [p..r]$  choice.

**Leftward** shift of elements  $\leq$  **pivot**  $x = A[r]$  and proper reposition of **pivot** next to them ensures that  $>$  elements are **rightward**.



Aug 2024

Algorithms and DS II: Randomization and Quicksort

5

5



PARTITION( $A, p, r$ )

```

 $x = A[r]$  // use last element for pivot
 $i = p - 1$  // initialize border of not larger elements
for  $j = p$  to  $r - 1$  // build  $A[p..q - 1]$ 
    if  $A[j] \leq x$  // not larger than pivot ?
        // yes, move element leftward
         $i = i + 1$  // shift border rightward
        exchange  $A[i]$  with  $A[j]$ 
    exchange  $A[i + 1]$  with  $A[r]$  // reposition pivot element
return  $i + 1$  // return pivot location

```

Aug 2024

Algorithms and DS II: Randomization and Quicksort

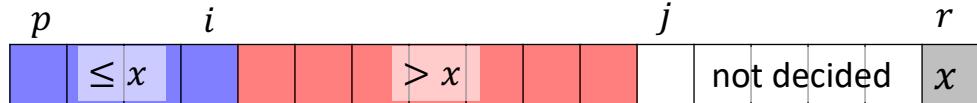
6

6

3



### Invariants of PARTITION



### Invariant correction proof.

At the beginning of the loop,  $\forall k$  there is

1. If  $p \leq k \leq i$ , then  $A[k] \leq x$ .
2. If  $i + 1 \leq k \leq j - 1$ , then  $A[k] > x$ .
3. If  $k = r$ , then  $A[k] = x$ .

The invariant holds prior to each iteration. Proof by induction on  $j$ .

Aug 2024

Algorithms and DS II: Randomization and Quicksort

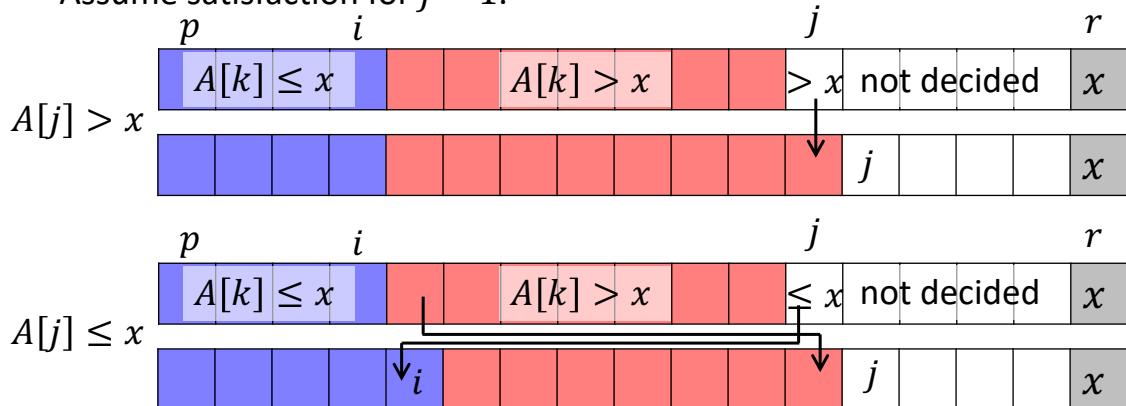
7

7



At first iteration  $i = p - 1$  and  $j = p$ ,  $\Rightarrow$  blue and red ranges are empty,  $\Rightarrow$  1 and 2 are trivially satisfied. First line of code satisfies 3.

Assume satisfaction for  $j - 1$ .



Aug 2024

Algorithms and DS II: Randomization and Quicksort

8

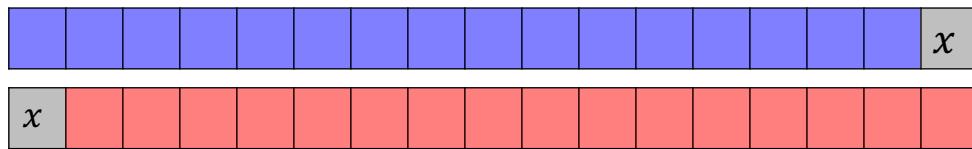
8



Upon termination  $j = r \Rightarrow 3$  is satisfied.

PARTITION runtime on  $A[p..r]$  is  $\Theta(r - p + 1)$ . (HW)

Worst-case PARTITION



QUICKSORT calls subproblems of size  $n - 1$  and 0.

$$T(n) = T(n - 1) + T(0) + \Theta(n) = T(n - 1) + \Theta(n) = \Theta(n^2).$$

(HW: prove  $T(n)$ , use substitution. What input yields worst-case?)

Aug 2024

Algorithms and DS II: Randomization and Quicksort

9

9



Best-case PARTITION



Results in  $\lfloor n/2 \rfloor$  and  $\lceil n/2 \rceil - 1$  subproblems. Ignoring  $\lfloor \cdot \rfloor$  and  $\lceil \cdot \rceil$ ,  $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$ . (HW: prove  $T(n)$ .)

For any fixed  $0 < \alpha \leq 1/2$ , if PARTITION always splits into  $\beta n$  and  $(1 - \beta)n - 1$ ,  $\alpha \leq \beta \leq 1/2$ ,  $T(n) = \Theta(n \log n)$ . (HW: prove.)

To overcome the worst-case input, pivot can be chosen randomly, expecting average partition to be reasonably balanced.

Aug 2024

Algorithms and DS II: Randomization and Quicksort

10

10



RANDOMIZED-PARTITION( $A, p, r$ )

```
i = RANDOM(p, r) // pivot drawn randomly
exchange A[r] with A[i] // make pivot rightmost
return PARTITION(A, p, r) // deterministic
```

RANDOMIZED-QUICKSORT( $A, p, r$ )

```
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q - 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Aug 2024

Algorithms and DS II: Randomization and Quicksort

11

11



### Run time analysis – method 1

Assume w.l.o.g that all  $A$ 's elements are different.

Let  $\Theta(n) = an$  be the runtime of RANDOMIZED-PARTITION, which may choose any of the  $n$  element with equal probability.  $\Rightarrow$

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

Multiplying (1) by  $n \Rightarrow$

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

Aug 2024

Algorithms and DS II: Randomization and Quicksort

12

12



Substitution  $n - 1$  into (2)  $\Rightarrow$

$$(3) (n - 1)T(n - 1) = a(n - 1)^2 + 2 \sum_{i=0}^{n-2} T(i).$$

Subtraction of (3) from (2)  $\Rightarrow$

$$\begin{aligned} nT(n) &= (n + 1)T(n - 1) + a(2n - 1) \\ &\leq (n + 1)T(n - 1) + 2a(n + 1). \end{aligned}$$

Division by  $n(n + 1)$   $\Rightarrow$

$$(4) T(n)/(n + 1) \leq T(n - 1)/n + 2a/n.$$

Aug 2024

Algorithms and DS II: Randomization and Quicksort

13

13



Substitution  $F(n) = T(n)/(n + 1)$  in (4)  $\Rightarrow$

$$\begin{aligned} (5) F(n) &\leq 2a/n + F(n - 1) \\ &\leq 2a/n + 2a/(n - 1) + F(n - 2) \\ &\leq 2a/n + 2a/(n - 1) + 2a/(n - 2) + F(n - 3) \leq \dots \end{aligned}$$

R.H.S of (5) summed to

$$F(n) \leq 2a \sum_{i=1}^n \frac{1}{i} = 2aH_n, \text{ where } H_n \text{ is the } n\text{th harmonic sum.} \Rightarrow$$

$$T(n) = 2a(n + 1)H_n = O(n \log n). \blacksquare$$

Aug 2024

Algorithms and DS II: Randomization and Quicksort

14

14



## Run time analysis – method 2

RANDOMIZED–PARTITION dominates runtime, called at most  $n$  times since element can be pivot at most once.

Within PARTITION loop there are comparisons  $A[j] \leq x$  of  $A[j]$  with the pivot  $x$ .

Let  $X$  be total number comparisons **over all** PARTITION's loops. Total QUICKSORT runtime is therefore  $O(n + X)$ .

So what is  $X$ ?

Aug 2024

Algorithms and DS II: Randomization and Quicksort

15

15



Rename the elements of  $A$  as  $z_1, z_2, \dots, z_n$  by their ascending sorted values and let  $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ .

How many times QUICKSORT compares  $z_i$  and  $z_j$ ,  $i \neq j$ ?

In PARTITION's comparison  $z \leq x$ ,  $x$  is the pivot, and once the loop completes  $x$  is placed in final location and **never touched again**.

Hence QUICKSORT compares any  $z_i$  and  $z_j$ ,  $i \neq j$ , at most once.

Aug 2024

Algorithms and DS II: Randomization and Quicksort

16

16



Define random variable  $X_{ij} = \begin{cases} 1 & z_i \text{ is compared to } z_j \\ 0 & \text{otherwise} \end{cases}$ .

Then,  $X = \sum_{1 \leq i < j \leq n} X_{ij} = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}$ , and

expectation linearity

$$(6) E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \Pr[z_i, z_j \text{ compared}].$$

RANDOMIZED-PARTITION chooses pivot  $x$  from subarray  $A[p..r]$  randomly and independently.

Aug 2024

Algorithms and DS II: Randomization and Quicksort

17

17

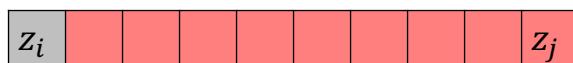


Let  $x \in Z_{ij}$ . There are 3 cases:

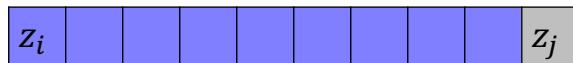
- $z_i < x < z_j$ .  $z_i$  and  $z_j$  never compared since  $x$  splits  $Z_{ij}$  into smaller and larger parts.



- $x = z_i$ .  $x$  is compared to all, hence  $z_i$  and  $z_j$  are compared.



- $x = z_j$ .  $x$  is compared to all, hence  $z_i$  and  $z_j$  are compared.



Aug 2024

Algorithms and DS II: Randomization and Quicksort

18

18



length of  $[z_i, \dots, z_j]$

$x$  chosen at random  $\Rightarrow \Pr[z_i \text{ is compared to } z_j] = 2/(j - i + 1)$ .

only  $x = z_i$  and  $x = z_j$  apply

Substitution into (6) yields

$$k = j - i$$

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n-i} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} <$$

$$\sum_{i=1}^{n-1} \sum_{k=1}^n \frac{2}{k} = 2 \sum_{i=1}^{n-1} H_n = O(n \log n). \blacksquare$$

harmonic sum

# מבנה נתונים ואלגוריתמים 2

## תירגול: Merge Sort

Based on: Introduction to Algorithms  
by Cormen, Leiserson, Rivest, and Stein

# Growth of Functions - $O$ , $\Omega$ , $\omega$ & $\Theta$ notations

- $f(n) \in O(g(n))$  **if there exist constants**  $c$  and  $n_0$  such that  $f(n) \leq c \cdot g(n)$  for all  $n \geq n_0$ 
  - Growth of  $f(n)$  is not faster than growth of  $g(n)$
- $f(n) \in \Omega(g(n))$  **if there exist constants**  $c$  and  $n_0$  such that  $f(n) \geq c \cdot g(n)$  for all  $n \geq n_0$ 
  - Growth of  $f(n)$  is not slower than growth of  $g(n)$
- $f(n) \in \Theta(g(n))$  if and only if  $f(n) \in O(g(n))$  and  $f(n) \in \Omega(g(n))$ 
  - Growth of  $f(n)$  is the same as growth of  $g(n)$

# Growth of Functions - $O, o, \Omega, \omega$ & $\Theta$ notations

- $f(n) \in o(g(n))$  **if for any constant**  $c$ , there exists a constant  $n_0$  such that  $f(n) < c \cdot g(n)$  for all  $n \geq n_0$ 
  - Growth of  $f(n)$  is strictly smaller than growth of  $g(n)$
- $f(n) \in \omega(g(n))$  **if for any constant**  $c$ , there exists a constant  $n_0$  such that  $f(n) > c \cdot g(n)$  for all  $n \geq n_0$ 
  - Growth of  $f(n)$  is strictly faster than growth of  $g(n)$

Common abuse of notation:

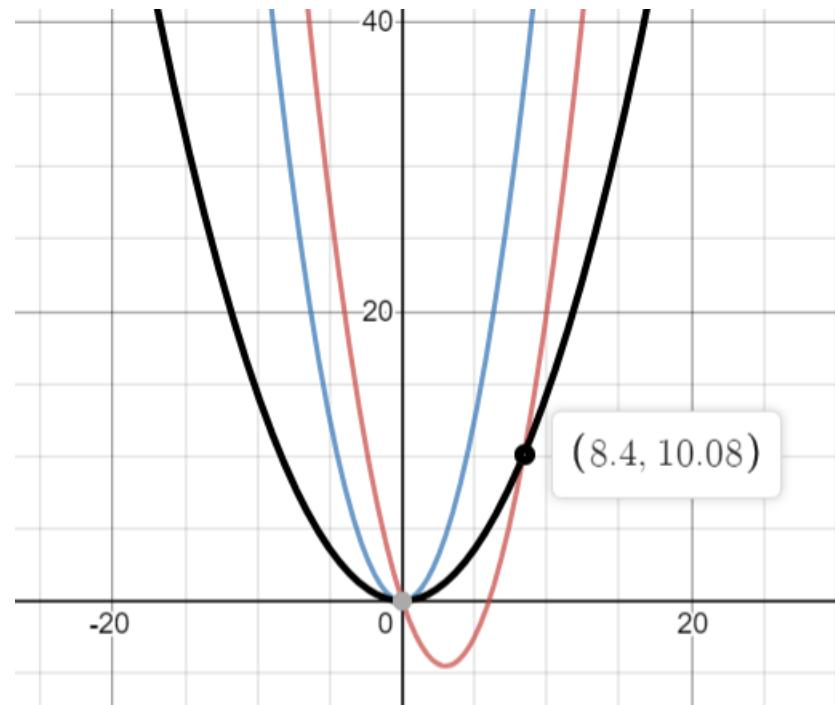
- We use  $f(n) = O(g(n))$  instead of  $f(n) \in O(g(n))$
- In the algorithmic literature people write  $f(n) = \Omega(g(n))$  instead of writing  $f(n) \notin o(g(n))$

# Growth of Functions - $O, o, \Omega, \omega$ & $\Theta$ notations

## Example:

- $0.5n^2 - 3n = \Theta(n^2)$  for  $c_1=1/7, c_2=1/2, n_0 = 8.4$

- $0.5n^2 - 3n$
- $(1/2)* n^2$
- $(1/7)* n^2$



# Example

- Question: what can we say about the following function?

$$f(n) = \begin{cases} n^2, & n \text{ is even} \\ 1, & n \text{ is odd} \end{cases}$$

- $f(n) \in O(n^2)$  since  $f(n) \leq n^2$ , for every  $n$ .
- $f(n) \notin O(1)$  since for any constant  $c$  there are infinitely many  $n$ 's for which  $f(n) > c \cdot 1$ .
- $f(n) \in \Omega(1)$  since  $f(n) \geq 1$ , for every  $n$ .
- $f(n) \notin \Omega(n^2)$  since for any constant  $c$  there are infinitely many  $n$ 's for which  $f(n) < c \cdot n^2$ .
- $f(n) \notin o(n^2)$  since for  $c = 1$  there are infinitely many  $n$ 's for which  $f(n) \geq c \cdot n^2$ .

# Divide and conquer approach

- Many algorithms are *recursive* in structure
- They typically follow a *divide-and-conquer* approach:
  1. **Divide** the problem into smaller sub problems.
  2. **Conquer** the sub problems by solving them recursively.
  3. **Combine** the solutions to the sub problems into the solution for the original problem.

# Algorithm Merge Sort

- Merge Sort is an efficient sort algorithm which follows the divide-and-conquer paradigm:
  1. **Divide:** Divide the  $n$ -element sequence to be sorted into two subsequences of  $(n/2)$  elements.
  2. **Conquer:** Sort the two subsequences recursively using merge sort.
  3. **Combine:** Merge the two sorted subsequences to produce the sorted answer.

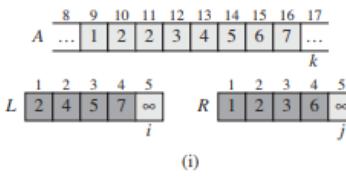
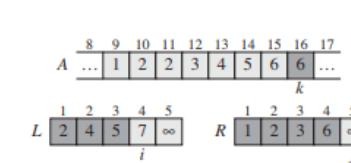
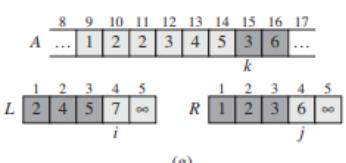
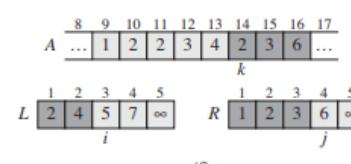
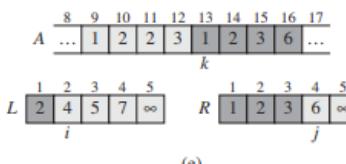
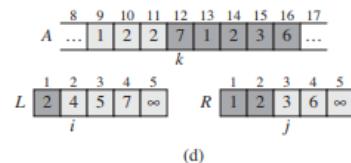
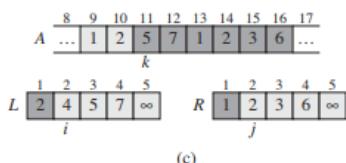
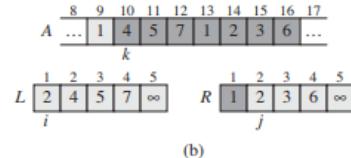
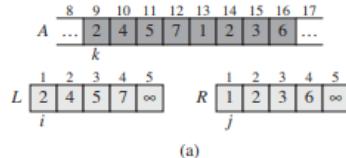
# Algorithm Merge Sort-pseudocode

```
MERGE-SORT( $A, p, r$ )
1  if  $p < r$ 
2     $q = \lfloor (p + r)/2 \rfloor$ 
3    MERGE-SORT( $A, p, q$ )
4    MERGE-SORT( $A, q + 1, r$ )
5    MERGE( $A, p, q, r$ )
```

- $A$  is n-element array to be sorted
- $p, q$  and  $r$  - indices into the array,  $p \leq q < r$
- Initial call to sort  $A$ :  $\text{Merge-Sort}(A, 1, n)$
- The recursion terminates when  $p=r \implies \text{length(sequence to be sorted)}=1$

# Algorithm Merge -pseudocode

- Merge is the “combine” step and key operation of the merge sort
- Merge procedure assumes that  $A[p..q]$  and  $A[q+1..r]$  are in sorted order



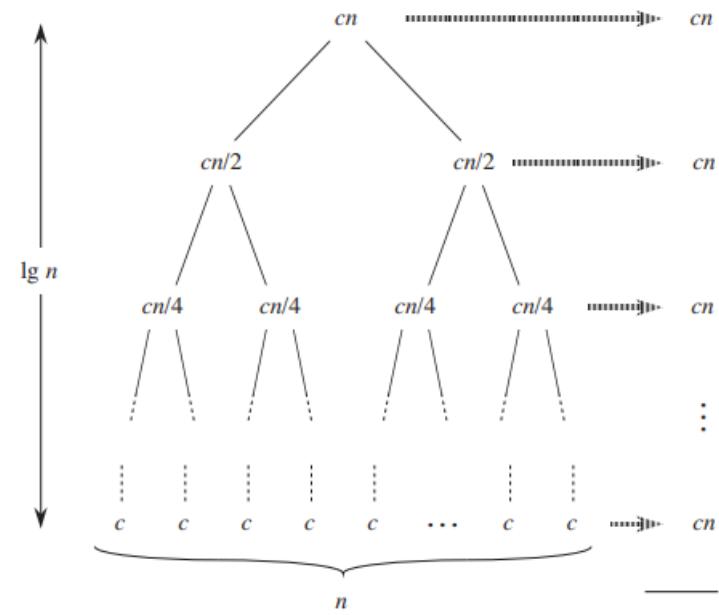
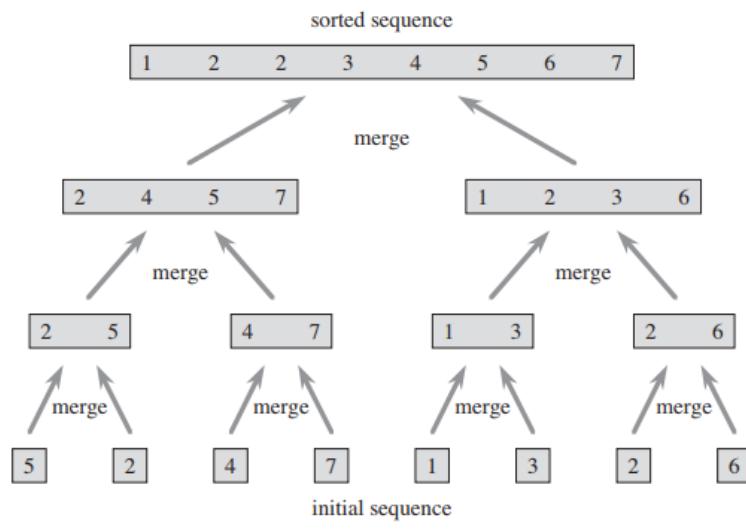
MERGE( $A, p, q, r$ )

```

1    $n_1 = q - p + 1$ 
2    $n_2 = r - q$ 
3   let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays
4   for  $i = 1$  to  $n_1$ 
5      $L[i] = A[p + i - 1]$ 
6   for  $j = 1$  to  $n_2$ 
7      $R[j] = A[q + j]$ 
8    $L[n_1 + 1] = \infty$ 
9    $R[n_2 + 1] = \infty$ 
10   $i = 1$ 
11   $j = 1$ 
12  for  $k = p$  to  $r$ 
13    if  $L[i] \leq R[j]$ 
14       $A[k] = L[i]$ 
15       $i = i + 1$ 
16    else  $A[k] = R[j]$ 
17       $j = j + 1$ 

```

# Algorithm Merge Sort - example



# Algorithm Merge - running time

MERGE( $A, p, q, r$ )

```
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays
4  for  $i = 1$  to  $n_1$ 
5       $L[i] = A[p + i - 1]$ 
6  for  $j = 1$  to  $n_2$ 
7       $R[j] = A[q + j]$ 
8   $L[n_1 + 1] = \infty$ 
9   $R[n_2 + 1] = \infty$ 
10  $i = 1$ 
11  $j = 1$ 
12 for  $k = p$  to  $r$ 
13     if  $L[i] \leq R[j]$ 
14          $A[k] = L[i]$ 
15          $i = i + 1$ 
16     else  $A[k] = R[j]$ 
17          $j = j + 1$ 
```

$c1$

$\Theta(n_1+n_2) = \Theta(n)$

$c2$

$\Theta(n_1+n_2) = \Theta(n)$

$\Theta(n)$

# Algorithm Merge Sort - running time

- Denote:  $T(n)$ - total running time of Merge Sort

MERGE-SORT( $A, p, r$ )

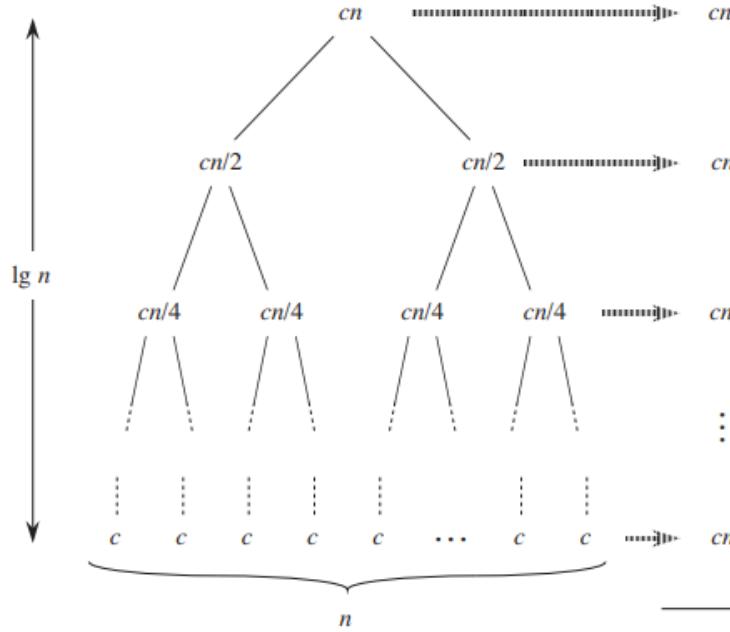
```
1  if  $p < r$ 
2     $q = \lfloor (p + r)/2 \rfloor$ 
3    MERGE-SORT( $A, p, q$ )
4    MERGE-SORT( $A, q + 1, r$ )
5    MERGE( $A, p, q, r$ )
```



$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases} \quad \longrightarrow \quad T(n) = \begin{cases} c & \text{if } n = 1, \\ 2T(n/2) + cn & \text{if } n > 1, \end{cases}$$

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + \Theta(n) = 2 * \left(2T\left(\frac{n}{4}\right) + \Theta\left(\frac{n}{2}\right)\right) + \Theta(n) = \dots = \\ &= \sum_{i=0}^{\log n} 2^i \Theta\left(\frac{n}{2^i}\right) = \Theta(n) \sum_{i=0}^{\log n} 1 = \Theta(n)(\log n + 1) = \Theta(n \log n) \end{aligned}$$

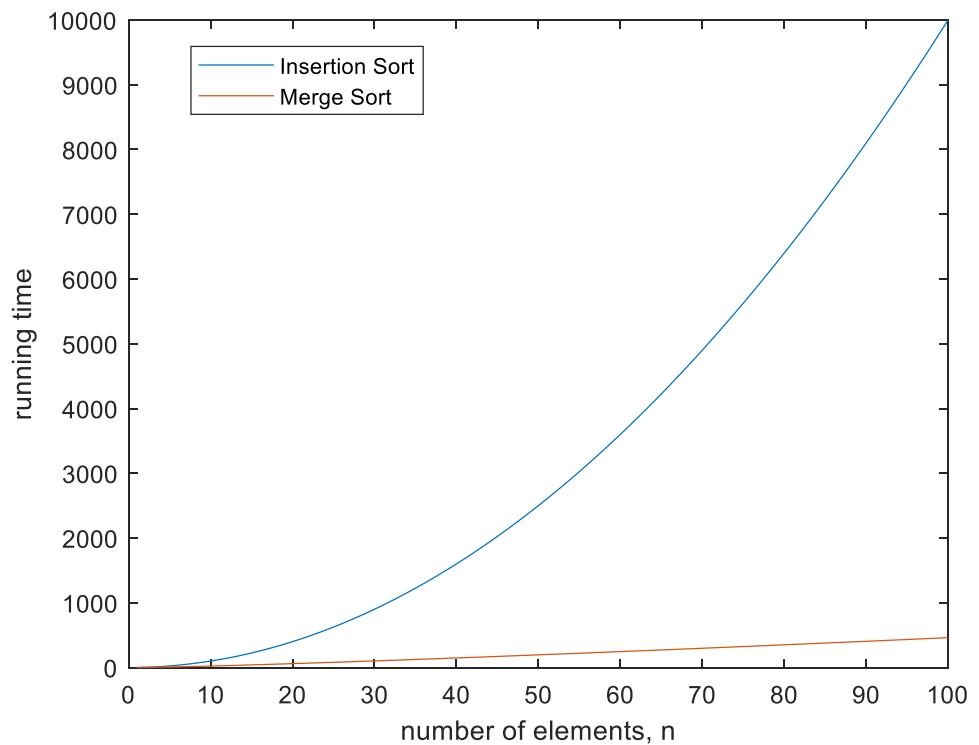
# Algorithm Merge Sort - recursion tree



- The total number of levels in the recursion tree:  $\log n + 1$  ,  
 $n$  = number of leaves = input size
- Running time at each level : at level  $i$  we have  $2^i$  nodes, while each contributes a cost of  $cn/2^i$ , so total cost =  $2^i cn/2^i = cn$
- Total running time:  $cn(\log n + 1) = cn \log n + cn = \Theta(n \log n)$

# Sort algorithms - comparison

- Merge Sort:  $\Theta(n \log n)$
- Insertion Sort:  $\Theta(n^2)$



# Algorithm Insertion Sort-reminder

INSERTION-SORT( $A$ )

```
1  for  $j = 2$  to  $A.length$ 
2      key =  $A[j]$ 
3      // Insert  $A[j]$  into the sorted sequence  $A[1 \dots j - 1]$ .
4       $i = j - 1$ 
5      while  $i > 0$  and  $A[i] > key$ 
6           $A[i + 1] = A[i]$ 
7           $i = i - 1$ 
8       $A[i + 1] = key$ 
```



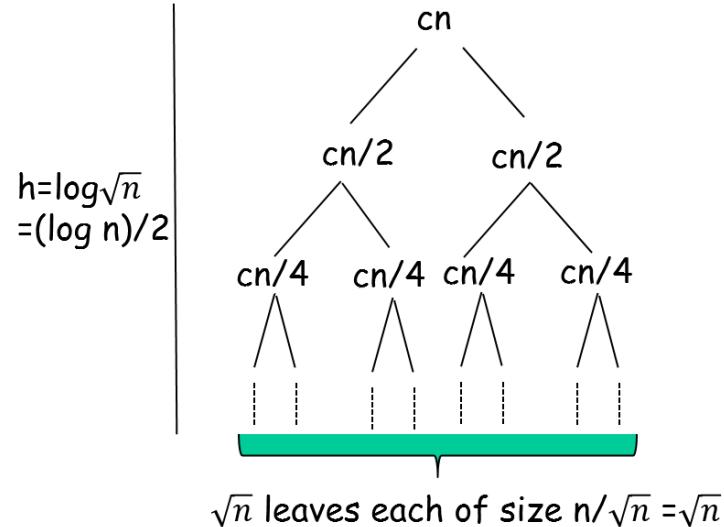
# Example 1

- Given the algorithm which combines Merge Sort with Insertion Sort:

```

MS-IS(A,p,r)
{
    if (r - p + 1 ≤ √size(A))
        then Insertion-Sort(A,p,r)
    else
    {
        q ← ⌊(r + p) / 2⌋
        MS-IS(A,p,q)
        MS-IS(A,q+1,r)
        Merge(A,p,q,r)
    }
}

```



- Show that running time of MS-IS is  $\Theta(n^{3/2})$ .

$$T(n) = \sum_{i=0}^{(\log n)/2} \Theta(n) + \sqrt{n} \Theta((\sqrt{n})^2) = \Theta(n \log n) + \Theta(n^{3/2}) = \Theta(n^{3/2})$$

$\uparrow$                      $\uparrow$   
 Merge                  IS

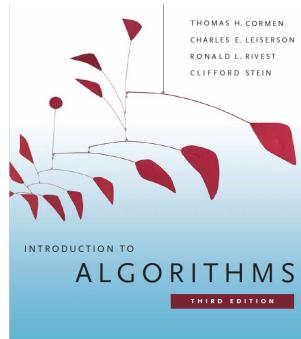
# Merge Sort Demo

- [https://www.youtube.com/watch?v=XaqR3G\\_NVoo](https://www.youtube.com/watch?v=XaqR3G_NVoo)



# Order Statistics

Prepared by Shmuel Wimer  
Courtesy of Prof. Dror Rawitz



Aug 2024

Algorithms and DS II: Order Statistics

1

1



## Order Statistics and Selection Problem

The ***i*th order statistics** of  $n$  elements is the  $i$ th smallest element.

Given a set  $A$  of  $n$  distinct elements, the  $i$ th order statistics **selection problem** is to find  $x \in A$  s.t.  $x$  is larger than exactly  $i - 1$  elements.

Sorting  $A$  in  $O(n \log n)$  time enables to solve in  $O(n)$  time, but faster solution in  $O(n)$  time (no sorting) is possible.

**Minimum**  $i = 1$  and **maximum**  $i = n$  problems complexity is  $\Theta(n)$ .

$\Theta(n)$  is possible for any  $i$ th order statistics.

Aug 2024

Algorithms and DS II: Order Statistics

2

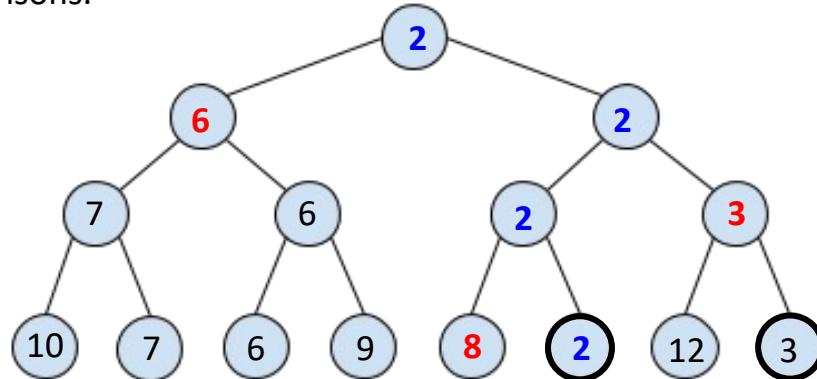
2

1



## Finding the 2<sup>nd</sup> Smallest

Can be found by knockout tournament with  $n + \lceil \log n \rceil - 2$  comparisons.



Aug 2024

Algorithms and DS II: Order Statistics

3

3



1. Divide input into  $k = \lceil n/2 \rceil$  element pairs.
2. Get the set of  $k$  smaller of every pair.
3. If  $k == 1$  stop, smallest found. Else set  $n = k$  and go back to 1.

Knockout tournament has  $n - 1$  comparisons.

Smallest wins after  $\lceil \log n \rceil$  comparisons, implying binary tree.

At some comparison **the smallest must beat** the 2<sup>nd</sup> smallest.

Traversal of the  $\lceil \log n \rceil - 1$  beats of smallest discover the 2<sup>nd</sup>.

Aug 2024

Algorithms and DS II: Order Statistics

4

4



## Selection of $k$ th Order Statistics in Expected $\Theta(n)$

A divide-and-conquer algorithm similar to Quicksort, except that it proceeds with only one of the partitions rather than both.

RANDOMIZED-SELECT seeks the  $i$ th order statistics of  $A$ .

It partitions the array recursively.

It chooses a pivot randomly with RANDOMIZED-PARTITION but proceed with only one side of the partition where  $i$  is surely located.

Aug 2024

Algorithms and DS II: Order Statistics

5

5



```
RANDOMIZED-SELECT( $A, p, r, i$ ) //  $i$ th order statistics
  if  $p == r$  return  $A[p]$  //  $A[p]$   $i$ th smallest element
    // partition  $A[p..r]$  into  $A[p..q - 1]$  and  $A[q + 1..r]$ 
    // around pivot  $A[q]$ 
     $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
     $k = q - p + 1$  // number of elements in  $A[p..q]$ 
    if  $i == k$  return  $A[q]$  //  $A[q]$  is  $i$ th smallest element
    else if  $i < k$  //  $i$ th is in the lower part
      return RANDOMIZED-SELECT( $A, p, q - 1, i$ )
    else //  $i$ th is in the higher part
      return RANDOMIZED-SELECT( $A, q + 1, r, i - k$ )
```

Aug 2024

Algorithms and DS II: Order Statistics

6

6



## Run time analysis

Worst-case run time is  $\Theta(n^2)$ . When it happens? (HW)

Run time  $T(n)$  on  $A[p..r]$  of  $n$  elements is random variable.

RANDOMIZED-PARTITION( $A, p, r$ ) likely returns any element as pivot  $A[q]$ .  $\Rightarrow \Pr[|[p..q]| = k] = 1/n, 1 \leq k \leq n$ .

Define random variable  $X_k = \begin{cases} 1 & |[p..q]| = k \\ 0 & \text{otherwise} \end{cases} \Rightarrow E[X_k] = \frac{1}{n}$ .

It is unknown a priori whether selection proceeds with  $A[p..q-1]$ ,  $A[q]$  or  $A[q+1..r]$ , so larger interval is safely assumed.

Aug 2024

Algorithms and DS II: Order Statistics

7

7



When  $X_k = 1$ , the implied subarrays have size  $k - 1$  and  $n - k$ .  $\Rightarrow$

$$T[n] \leq \sum_{k=1}^n X_k \cdot (T(\max(k-1, n-k)) + O(n)).$$

Taking expectation and applying its linearity, there is

$$E[T[n]] \leq \sum_{k=1}^n E[X_k \cdot T(\max(k-1, n-k))] + O(n)$$

Independence:  $\Pr[X_k = 0 / 1 | \max(k-1, n-k)] = \Pr[X_k = 0 / 1]$

$$= \sum_{k=1}^n E[X_k] \cdot E[T(\max(k-1, n-k))] + O(n)$$

$$= \frac{1}{n} \sum_{k=1}^n E[T(\max(k-1, n-k))] + O(n).$$

Aug 2024

Algorithms and DS II: Order Statistics

8

8



There is  $\max(k - 1, n - k) = \begin{cases} k - 1 & \text{if } k > \lceil n/2 \rceil \\ n - k & \text{if } k \leq \lceil n/2 \rceil \end{cases}$

In  $\sum_{k=1}^n E[\cdot]$ , for even  $n$  each term from  $T(\lceil n/2 \rceil)$  to  $T(n - 1)$  appears twice. Same for odd  $n$  plus  $T(\lfloor n/2 \rfloor)$  which appears once.  $\Rightarrow$

$$E[T[n]] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + O(n).$$

Above recursion is solved by substitution, yielding  $E[T(n)] = O(n)$ .  
(HW, see CLRS 9.2).

Aug 2024

Algorithms and DS II: Order Statistics

9

9



## Selection of $k$ th Order Statistics in $O(n)$ Worst Case

$\text{SELECT}(A, k)$

1. Divide input into  $\left\lceil \frac{n}{5} \right\rceil$  groups of 5 elements, one possibly smaller.
2. Find the median of each group. Let  $B$  be the set of all medians.
3. Let  $x = \text{SELECT}\left(B, \left\lfloor \frac{|B|}{2} \right\rfloor\right)$  be  $B$ 's median.
4. Divide  $A$  around  $x$ :  $L = \{a \in A \mid a < x\}$ ,  $R = \{a \in A \mid a > x\}$ .
5. If  $k = |L| + 1$  return  $x$ .  $k > |L| + 1$
6. If  $k \leq |L|$  then  $\text{SELECT}(L, k)$  else  $\text{SELECT}(R, k - (|L| + 1))$ .

Aug 2024

Algorithms and DS II: Order Statistics

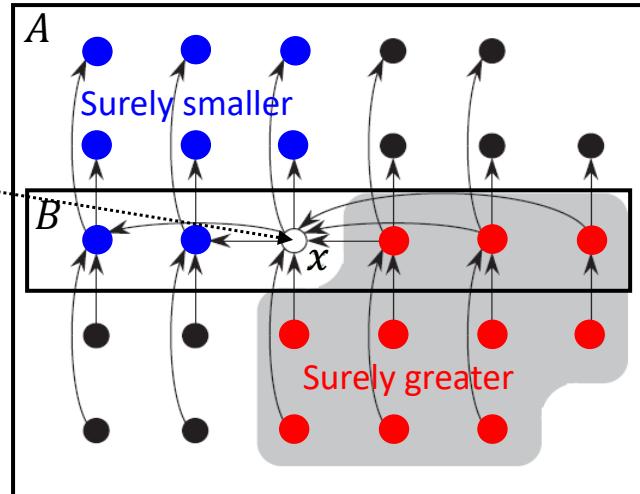
10

10



$\bullet \rightarrow \bullet$   
greater to smaller  
 $x: \lfloor |B|/2 \rfloor$

$|L| \geq \frac{n}{4}$  and  $|R| \geq \frac{n}{4}$   
guaranteed.



Aug 2024

Algorithms and DS II: Order Statistics

11

11

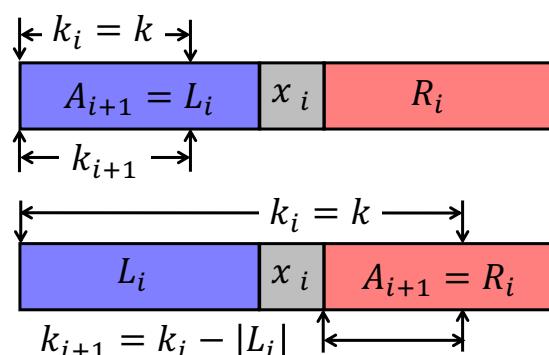
### Correctness proof:

Consider  $\text{SELECT}(A_i, k_i)$  call at line 6, where  $A_0 = A$  and  $k_0 = k$ .  
Assume by induction that  $k_i = k$  and consider  $(A_{i+1}, k_{i+1})$ . Then

If  $k_i \leq |L_i|$

then  $k_{i+1} = k_i = k$

else  $k_{i+1} = k_i - (|L_i| + 1)$



Aug 2024

Algorithms and DS II: Order Statistics

12

12



**Run time analysis:** Assume w.l.o.g that all the elements are distinct.

Then at least half of the medians in step 2 are not smaller than  $x$ .

$\Rightarrow$  At least half of the  $\left\lceil \frac{n}{5} \right\rceil$  groups contribute 3 elements  $> x$ , maybe except one smaller group and the group containing  $x$ .

$$\Rightarrow |\textcolor{red}{R}| \geq 3 \left( \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil - 2 \right) \geq \frac{3n}{10} - 6. \text{ Similarly, } |\textcolor{blue}{L}| \geq \frac{3n}{10} - 6.$$

$$\text{Consequently, } \frac{3n}{10} - 6 \leq |\textcolor{blue}{L}|, |\textcolor{red}{R}| \leq \frac{7n}{10} + 6.$$

Aug 2024

Algorithms and DS II: Order Statistics

13

13



Let  $T(n)$  be the worst-case run time. Steps 1, 2 and 4 take  $O(n)$  time.

Steps 3 takes  $T(\lceil n/5 \rceil)$  time.

Step 6 takes at most  $T(7n/10 + 6)$  time.

The following recurrence is in order

$$T(n) = T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n).$$

We show that  $T(n) = O(n)$ .

Assume  $T(n) \leq cn$  for sufficiently large  $n$  and a constant  $c$ . Then

Aug 2024

Algorithms and DS II: Order Statistics

14

14



$$T(n) \leq c(\lceil n/5 \rceil) + c(7n/10 + 6) + an \quad O(n) = an$$

$$\leq c(n/5 + 1) + c(7n/10 + 6) + an$$

$$= 9cn/10 + 7c + an \quad \text{by assumption}$$

$$= cn + (-cn/10 + 7c + an) \leq cn$$

$$\Rightarrow -cn/10 + 7c + an \leq 0$$

$$\Rightarrow (1) \ c \geq 10a(n/(n-70)) \rightarrow 10a \text{ as } n \rightarrow \infty.$$

Choosing  $c \geq 20a$  will satisfy (1). ■

Aug 2024

Algorithms and DS II: Order Statistics

15

15



What happens if 5 is replaced by 3? 7? (HW).

SELECT is a deterministic algorithm. We can use probabilistic pivot choice in line 3 (as in Quicksort) and then proceed in one side of the partitions.

Show that its expected run time is linear (HW).

Quicksort can use SELECT for balanced partition, yielding worst-case  $O(n \log n)$  time deterministic Quicksort.

Impractical because of the large constants in SELECT.

Aug 2024

Algorithms and DS II: Order Statistics

16

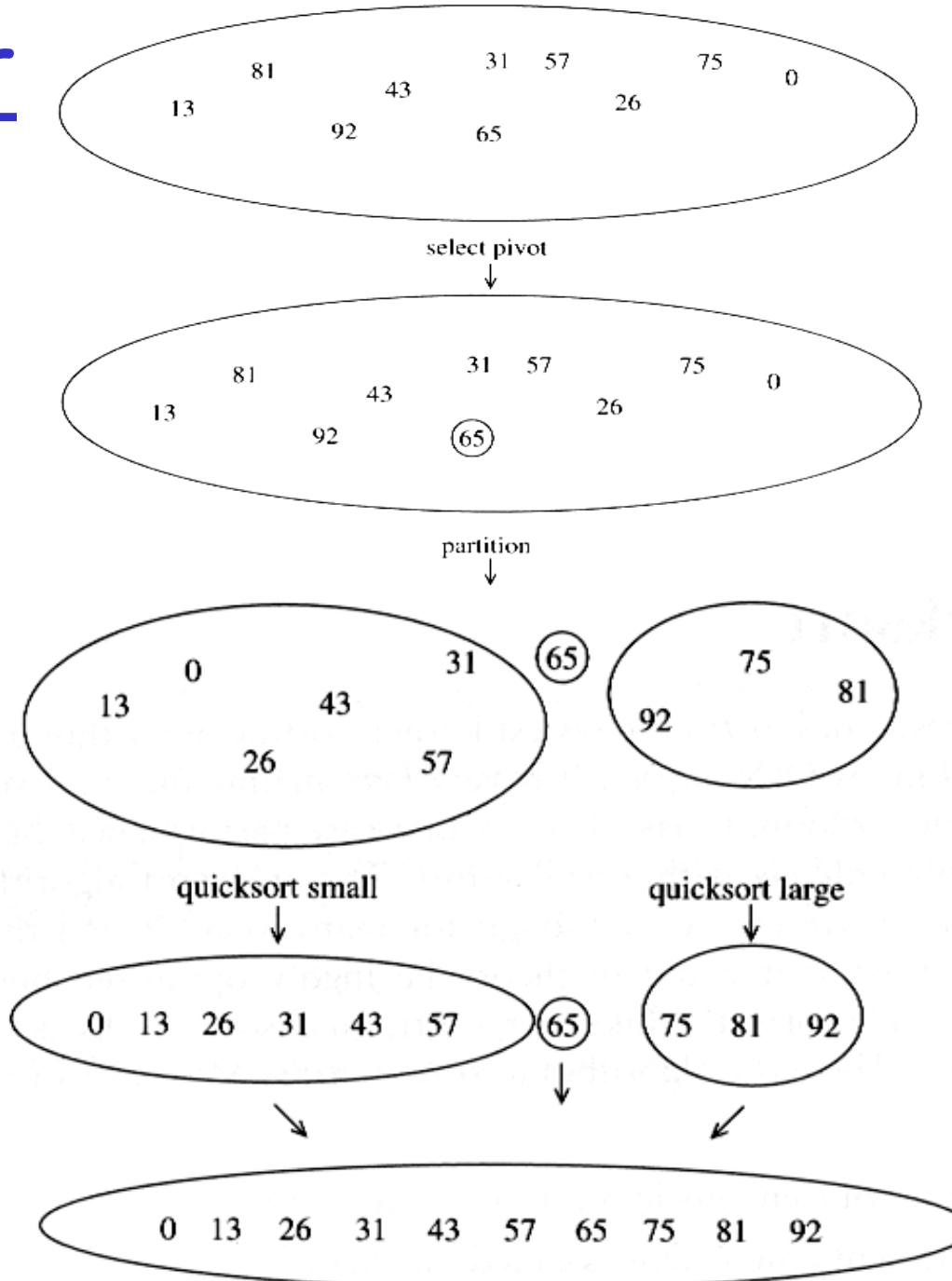
16

# מבנה נתונים ואלגוריתמים 2

# תירגול: Quicksort and Order Statistics

Based on: Introduction to Algorithms  
by Cormen, Leiserson, Rivest, and Stein

# Quicksort Example



# Quicksort - Pseudocode

**Quicksort( $A, p, r$ ):**

- If  $p < r$ , then
  - $q \leftarrow \text{Partition}(A, p, r)$
  - $\text{Quicksort}(A, p, q - 1)$
  - $\text{Quicksort}(A, q + 1, r)$

**Partition( $A, p, r$ ):**

- $x \leftarrow A[r]$
- $i \leftarrow p - 1$
- For  $j \leftarrow p$  to  $r - 1$  do
  - If  $A[j] \leq x$  then
    - $i \leftarrow i + 1$
    - $A[i] \leftrightarrow A[j]$
- $A[i + 1] \leftrightarrow A[r]$
- Return  $i + 1$

קריאה התחלתית:

$\text{Quicksort}(A, 1, n)$

הערה: האיבר  $x$  שנבחר ע"י השגרה נקרא **פיבוט** (pivot).

# Worst-case partitioning

## □ The worst-case behavior

- The pivot is the smallest element, all the time  
(always produced two subproblem: with  $n - 1$  elements and with 0 elements)
- Partition is always unbalanced
- The partitioning costs  $\Theta(n)$ ,  $T(1) = \Theta(1)$
- $T(n) = T(n - 1) + T(0) + \Theta(n) = T(n - 1) + \Theta(n)$

$$T(N) = T(N - 1) + cN$$

$$T(N - 1) = T(N - 2) + c(N - 1)$$

$$T(N - 2) = T(N - 3) + c(N - 2)$$

⋮

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

$$T(N) = T(1) + c \sum_{i=2}^N i = O(N^2)$$

# Best-case partitioning

## □ What will be the best case?

- Pivot is always in the middle (median of the array)
- Partition is perfectly balanced (each time produced two subarrays each of size no more than  $\frac{n}{2}$ )
- $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$

$$T(N) = 2T(N/2) + cN$$

$$\frac{T(N)}{N} = \frac{T(N/2)}{N/2} + c$$

$$\frac{T(N/2)}{N/2} = \frac{T(N/4)}{N/4} + c$$

$$\frac{T(N/4)}{N/4} = \frac{T(N/8)}{N/8} + c$$

⋮

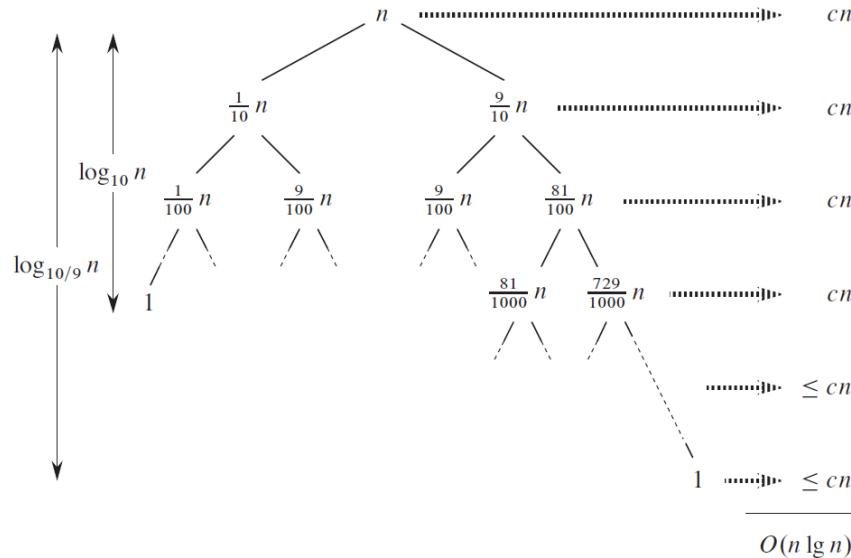
$$\frac{T(2)}{2} = \frac{T(1)}{1} + c$$

$$\frac{T(N)}{N} = \frac{T(1)}{1} + c \log N$$

$$T(N) = cN \log N + N = O(N \log N)$$

# Balanced partition - example

- A recursion tree for QUICKSORT in which PARTITION always produces a 9-to-1 split



- The longest simple path from the root to a leaf is  $n \rightarrow (\frac{9}{10})n \rightarrow (\frac{9}{10})^2n \rightarrow \dots 1$
- Height of the tree  $h = \log_{10/9} n$
- $(\frac{9}{10})^h n = 1 \rightarrow n = (\frac{10}{9})^h \rightarrow \log_{10/9} n = h \cdot \log_{10/9} \frac{10}{9} = h$
- Total time is smaller than  $cn \log_{10/9} n = ? O(n \lg n)$
- $n \log_{10/9} n < cn \log n \rightarrow n \frac{\log n}{\log 10/9} < cn \log n \rightarrow \frac{1}{\log 10/9} < c \rightarrow c > 6.6$
- Total time is  $O(n \lg n)$

## תזכורת מהרצאה:

### מציאת סטטיסטי הסדר ה- $k$

- נרצה לטפל במקרה הכללי של מציאת סטטיסטי הסדר ה-  $k$ .
- האלגוריתם משתמש בטכנית **הפרד ומשול**.
- האלגוריתם משתמש ב**פיבוט** כדי לחלק את המערך לשני חלקים, אבל בניגוד ל- Quicksort הוא ימשיך לטפל רק בחלק אחד.
- לכן נקבל זמן סיבוכיות  $(n)^{\Theta}$ .

# אלגוריתם למציאת סט' הסדר ה- $k$

אלגוריתם ( $Select(A,k)$ ):

- .1 אם  $A$  קטן מספיק אז פטור ע"י מילוי.
- .2 חלк את הקלט לחמישיות.
- .3 מצא חציו בכל חמישייה. נסמן את קבוצת החציוונים ע"י  $B$ .
- .4 מצא את החציוון של  $B$  שיסומן ע"י  $x$ , כלומר בצע  $\left(\left\lceil \frac{|B|}{2} \right\rceil\right)$
- .5 השתמש ב-  $x$  בפיבוט: נחלק את  $A$  לשני חלקים. נסמן ב-  $A'$  את קבוצת המספרים שגדולים מ-  $x$ .
- .6 אם  $|A' \setminus A'| \leq k$  אז נקרא ל-  $Select(A \setminus A',k)$
- אם  $|A' \setminus A'| > k$  אז נקרא ל-  $Select(A',k - |A \setminus A'|)$

הערה: חסרים פרטים בסיס הרקוריוסיה.

## נכונות

טענה: האלגוריתם מוצא את סטטיסטי הסדר ה-  $k$  במערך  $A$ .

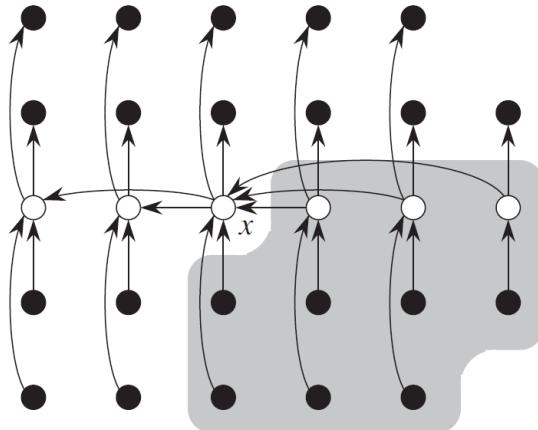
הוכחה:

- נסמן ב-  $A_i$  וב-  $k_i$  את הקלט לקריאה הרקורסיבית ה-  $i$ . נוכיח באינדוקציה שסטטיסטי הסדר ה-  $k$  במערך  $A$  הוא סטטיסטי הסדר ה-  $k_i$  במערך  $A_i$ .
- בסיס: ברור שזה נכון עבור  $k_1, A_1$ .

צעד: □

- נניח שהטענה נכונה עבור  $A_{i-1}, k_{i-1}$ .
  - אם  $|A'_{i-1} \setminus A_{i-1}| \leq k_{i-1}$ , אז סטטיסטי הסדר ה-  $k_{i-1}$  במערך  $A_{i-1}$  קטן שווה מהפיבוט  $x$ , ולכן סטטיסטי הסדר ה-  $k_{i-1}$  במערך  $A_{i-1}$  שווה לסטטיסטי הסדר ה-  $k_i = k_{i-1} \setminus A'_{i-1}$  במערך  $A_i = A_{i-1} \setminus A'_{i-1}$ .
  - אם  $|A'_{i-1} \setminus A_{i-1}| > k_{i-1}$ , אז סטטיסטי הסדר ה-  $k_{i-1}$  במערך  $A_{i-1}$  גדול מהפיבוט  $x$ , ולכן סטטיסטי הסדר ה-  $k_{i-1}$  במערך  $A_{i-1}$  שווה לסטטיסטי הסדר ה-  $|A'_{i-1} \setminus A_{i-1}| - k_{i-1} = k_i$  במערך  $A_i = A'_{i-1}$ .
- מסקנה: נכונות האלגוריתם נובעת מהנכונות של אלגוריתם המיאן.

## ניתוח זמן ריצה



טענה:  $|A'| \geq \frac{3}{10}n - 6$

הוכחה:

□ ישנו  $\left\lceil \frac{n}{5} \right\rceil$  חמישיות.

□ בכל הקבוצות יש חמישה אברים, מלבד אולי קבוצה אחת שמכילה פחות מ- 5 אברים (אם  $n$  לא מתחלק ב- 5).

□ לפחות חצי מהחצינאים גדולים או שווים לחציון החצינאים  $x$ . אך לפחות חצי מהקבוצות תורמות 3 אברים שגדולים מ-  $x$ , למעט הקבוצה של  $x$ , ואולי קבוצה שמכילה פחות מ- 5 אברים.

סה"כ:  $|A'| \geq 3 \left( \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil - 2 \right) \geq \frac{3}{10}n - 6$

טענה:  $|A \setminus A'| \geq \frac{3}{10}n - 6$

הוכחה: דומה.

מסקנה:  $\frac{3}{10}n - 6 \leq |A'|, |A \setminus A'| \leq \frac{7}{10}n + 6$

## ניתוח זמן ריצה (המשך)

- נסמן את זמן הריצה של האלגוריתם עם קלט באורך  $a$  ע"י  $T(n)$ .
- צעדים 2, 3, ו- 5 לוקחים זמן  $O(n)$ , נניח  $a \cdot a$ , עבור קבוע  $a$  כלשהו.
- צעד 4 לוקח זמן  $\left(\left\lceil \frac{n}{5} \right\rceil\right) \cdot T$ .
- צעד 6 לוקח זמן  $\left(\frac{7}{10}n + 6\right) \cdot T$ .
- סה"כ קיבלנו:

$$T(n) \leq \begin{cases} T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7}{10}n + 6\right) + an, & n > 140 \\ O(1), & n \leq 140 \end{cases}$$

- נראה שזמן הריצה הוא לינארי בשיטת הצבה
- נניח שזמן הריצה  $n \cdot c \leq T(n)$ , עבור קבוע  $c$  כלשהו.
- ברור שזה נכון עבור  $140 \leq n$  כי אפשר להשתמש באלגוריתם מיון.
- נסמן את הקבוע הרלוונטי ב-  $d$ .

## ניתוח זמן ריצה (המשך)

□ אחרית,

$$\begin{aligned}
 T(n) &\leq T\left(\frac{n}{5} + 1\right) + T\left(\frac{7}{10}n + 6\right) + an \leq c\left(\frac{n}{5} + 1\right) + c\left(\frac{7}{10}n + 6\right) + an \\
 &\leq c\frac{9}{10}n + 7c + an = cn + \left[7c + an - \frac{c}{10}n\right]
 \end{aligned}$$

□ נרצה שיתקיים

$$\begin{aligned}
 7c + an - \frac{c}{10}n &\leq 0 \\
 c\left(\frac{n}{10} - 7\right) &\geq an \\
 c &\geq 10a \cdot \frac{n}{n - 70}
 \end{aligned}$$

□ אם  $140 > n$  נקבל

$$c \geq 20a$$

□ לכן אם השתמש ב-  $c = \max\left\{20a, \frac{b}{140}\right\}$

# Exercise 9.3 from CLRS

- In the algorithm SELECT, the input elements are divided into groups of 5.
- Will the algorithm work in linear time if they are divided into groups of 7?
- Argue that SELECT does not run in linear time if groups of 3 are used.

# Answer part 1

- Groups of 7:

- Number of elements greater than  $x$ :

$$4 * \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{7} \right\rceil \right\rceil - 2 \right) \geq \frac{4n}{14} - 8$$

- Number of elements less or equal  $x$ :

$$n - \frac{4n}{14} + 8 = \frac{10n}{14} + 8$$

- We can show by substitution that the following expression is linear:

$$T(n) \leq T(n/7) + T(5n/7 + 8) + O(n)$$

- Suppose  $T(n) < cn$  for  $n < k$ , then, for  $m \geq k$ :

$$T(m) \leq T(m/7) + T(5m/7 + 8) + O(m) \leq c(m/7) + c(5m/7 + 8) + am \leq (6/7)cm + am$$

$$T(m) \leq cm + (-cm/7 + am)$$

$$(-cm/7 + am) < 0 \text{ for } c > 7a$$

$$T(n) = O(n)$$

- The algorithm will work for every odd number greater than 5.

## Answer part 2

- Groups of 3:

- Number of elements greater than x:

$$2 * \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{3} \right\rceil \right\rceil - 2 \right) \geq \frac{2n}{6} - 4$$

- Number of elements less or equal x:

$$n - \frac{2n}{6} + 4 = \frac{4n}{6} + 4$$

- The recurrence we get is :

$$T(n) = T\left(\left\lceil \frac{n}{3} \right\rceil\right) + T(2n/3+4) + O(n) \leq T(n/3) + T(2n/3) + O(n)$$

- We will show that  $O(n \log n)$  is an upper bound for the solution to this recurrence.

## Answer part 2

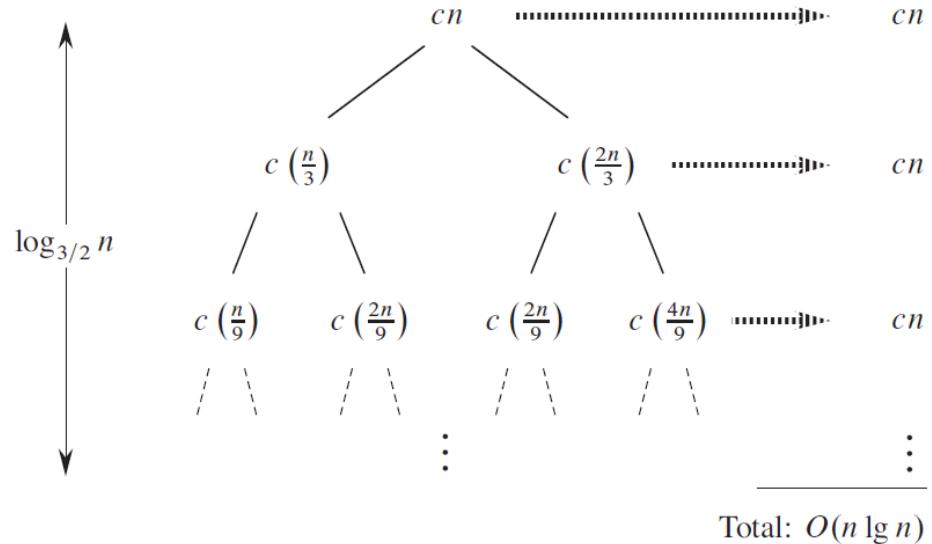
- We show that  $T(n) \leq d \cdot n \cdot \log n$ , where  $d > 0$

$$\begin{aligned} T(n) &\leq T(n/3) + T(2n/3) + cn \\ &\leq d(n/3) \lg(n/3) + d(2n/3) \lg(2n/3) + cn \\ &= (d(n/3) \lg n - d(n/3) \lg 3) \\ &\quad + (d(2n/3) \lg n - d(2n/3) \lg(3/2)) + cn \\ &= dn \lg n - d((n/3) \lg 3 + (2n/3) \lg(3/2)) + cn \\ &= dn \lg n - d((n/3) \lg 3 + (2n/3) \lg 3 - (2n/3) \lg 2) + cn \\ &= dn \lg n - dn(\lg 3 - 2/3) + cn \\ &\leq dn \lg n , \end{aligned}$$

- It grows more quickly than linear:  $T(n) = O(n \log n)$

# Answer part 2 - recursion tree

- A recursion tree for  $T(n) = T(n/3) + T(2n/3) + O(n)$

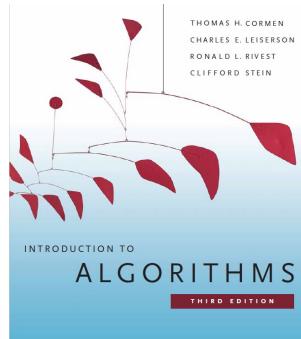


- The longest simple path from the root to a leaf is  $n \rightarrow (\frac{2}{3})n \rightarrow (\frac{2}{3})^2 n \rightarrow \dots 1$
- Height of the tree  $k = \log_{3/2} n$
- Total time is smaller than  $cn \log_{3/2} n = ? O(n \log n)$
- $n \log_{3/2} n < cn \log n \rightarrow n \frac{\log n}{\log 3/2} < cn \log n \rightarrow \frac{1}{\log 3/2} < c \rightarrow c > 1.7$
- Total time is  $O(n \log n)$



# Disjoint Sets

Prepared by Shmuel Wimer  
Courtesy of Prof. Dror Rawitz



September 2024

Algorithms and DS II: Disjoint Sets

1

1



# Disjoint-Set Operations

$N = \{1, 2, \dots, n\}$  grouped in dynamic sets  $\mathcal{S} = \{S_1, S_2, \dots, S_k\}$  s.t.

$\forall i \ S_i \subseteq N, \ \forall i \neq j \ S_i \cap S_j = \emptyset. \ r_i \in S_i$  is called **representative**.

Three operations are supported

**MAKE( $x$ )**: for  $x \in N$  creates  $S_x, |S_x| = 1, r_x = x$ .

**FIND( $z$ )**: returns  $r_x \in S_x$  if  $z \in S_x$ , or  $\emptyset$  if  $z \notin N$ .

**UNION( $z, w$ )**: returns  $S_x \cup S_y, r_x = \text{FIND}(z)$  and  $r_y = \text{FIND}(w)$ , with new representative ( $r_x$  or  $r_y$ ).  $S_x$  and  $S_y$  are removed from  $\mathcal{S}$ .

September 2024

Algorithms and DS II: Disjoint Sets

2

2

1



**Example:**  $N = \{1, 2, \dots, 8\}$ .  $\mathcal{S}: S_2 = \{\textcolor{blue}{2}, 6\}, S_4 = \{\textcolor{blue}{4}, 8\}, S_7 = \{1, 3, \textcolor{blue}{7}\}$ .

$\text{FIND}(8) = 4$  since  $8 \in S_4$  and  $r_4 = 4$ .

$\text{UNION}(2, 4) \Rightarrow \mathcal{S}: S_2 = \{\textcolor{blue}{2}, 4, 6, 8\}, S_7 = \{1, 3, \textcolor{blue}{7}\}$ .

$\text{MAKE}(5) \Rightarrow \mathcal{S}: S_2 = \{\textcolor{blue}{2}, 4, 6, 8\}, S_5 = \{\textcolor{blue}{5}\}, S_7 = \{1, 3, \textcolor{blue}{7}\}$ .

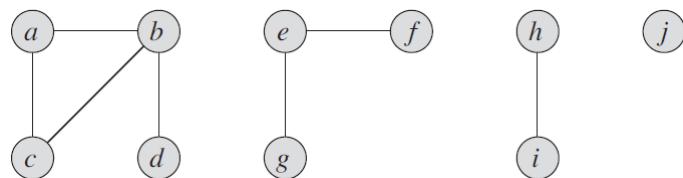
$\text{FIND}(8) = 2$  since  $8 \in S_2$  and  $r_2 = 2$ .

There are at most  $n$  MAKE and  $n - 1$  UNION operations.

We consider a series of total  $m$  MAKE, UNION and FIND operations.



**Example:** Find the connected components of simple graph  $G(V, E)$ .



$V: \{a, b, c, d, e, f, g, h, i\}$

$E: \{(a, b), (a, c), (b, d), (e, f), (e, g), (h, i)\}$

$G$ 's DS can be adjacency matrix, edge list, or alike.



**CONNECTED-COMPONENT( $G(V, E)$ )**

$\forall v \in V$  MAKE( $v$ ) // initially every set is a single vertex

$\forall (u, v) \in E$

// check if edge's vertices already in same component

if FIND( $u$ )  $\neq$  FIND( $v$ )

UNION( $u, v$ ) // merge two component

Initially:  $S_a = \{a\}, S_b = \{b\}, \dots, S_j = \{j\}$ .

Output:  $S_a = \{a, b, c, d\}, S_e = \{e, f, g\}, S_h = \{h, i\}, S_j = \{j\}$ .

**SAME-COMPONENT( $u, v$ )**

return FIND( $u$ ) == FIND( $v$ )

September 2024

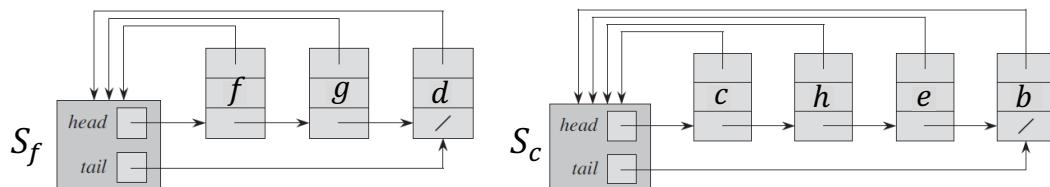
Algorithms and DS II: Disjoint Sets

5

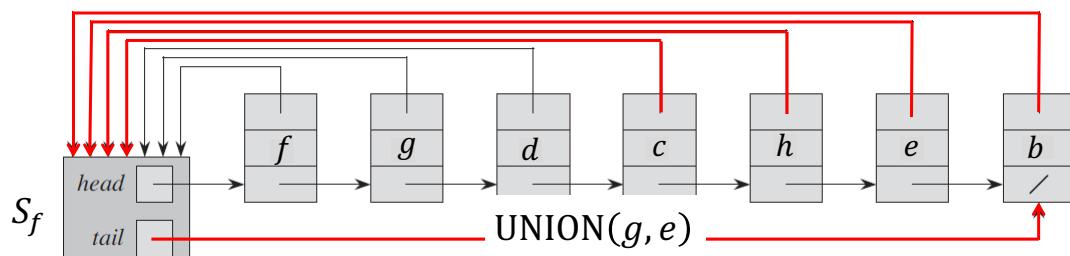
5



## Linked List Representation



MAKE and FIND require  $O(1)$  time.



Naive UNION implementation takes  $O(n)$  due to pointer updates.

September 2024

Algorithms and DS II: Disjoint Sets

6

6



operation	# updates	
MAKE( $x_1$ )	1	Given $n$ objects $\{x_1, x_2, \dots, x_n\}$ , the
:	:	following $2n - 1$ operation
MAKE( $x_n$ )	1	sequence results in
UNION( $x_2, x_1$ )	1	$\sum_{i=1}^n 1 + \sum_{i=1}^{n-1} i = \Theta(n^2)$
:		
UNION( $x_i, x_{i-1}$ )	$i-1$	total number of object updates,
:		hence $\Theta(n^2)$ run time, with $\Theta(n)$
UNION( $x_n, x_{n-1}$ )	$n-1$	amortized time per operation.



## Weighted-Union Heuristic

**Weighted-union heuristic** append always shorter list to longer, resulting in fewer object updates.

Each set stores its size, which upon UNION is  $O(1)$  time to update.

**Theorem 1:** Using linked list DS with weighted-union heuristic, sequence of total  $m$  MAKE, UNION, and FIND,  $n$  of which are MAKE, takes  $O(m + n \log n)$  time.

**Proof:** Consider the pointer updates of an object  $x$  along the entire sequence of  $m$  operations.



UNION updates  $x$ 's pointer only if it belongs to a set not larger than the counterpart, case where  $x$ 's set size is at least **doubled**.

Since there are  $n$  objects, size doubling (and hence pointer update) can occur at most  $\lceil \log n \rceil$  times.

Hence UNION time is bounded by  $O(n \log n)$  plus  $O(n)$  due to list concatenations.

Since each MAKE and FIND takes  $O(1)$ , the time for the entire sequence of  $m$  operations is  $O(m)$ .

The total time is thus  $O(m + n \log n)$ . ■



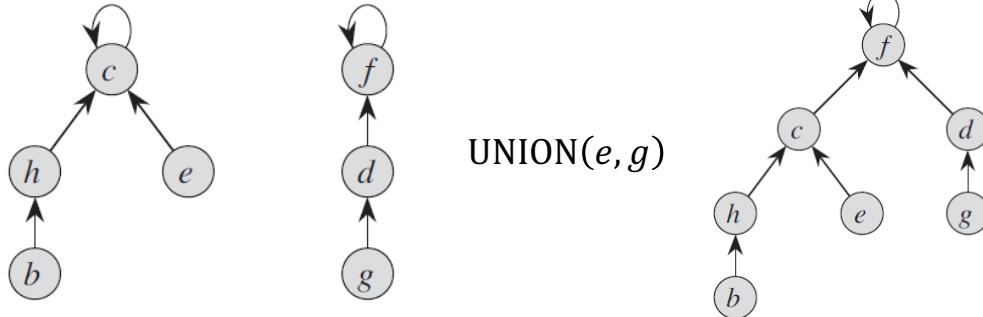
## Disjoint-Set Forests

Run time speedup is achieved by rooted trees, with each node containing one member and each tree representing one set.

Each member points only to its parent. The root contains the set's representative and it is its own parent.

The speedup is enabled by two heuristics: **union by rank** and **path compression**.

**Disjoint-set forests** is asymptotically optimal disjoint-set DS, achieving almost linear run-time (shown later).



MAKE creates one-node tree, setting its rank to 0.

FIND follows pointers up to root.

UNION causes pointer of one tree to point to other's root. Poor implementation may create a tree that is linear chain of  $n$  nodes.

September 2024

Algorithms and DS II: Disjoint Sets

11

11



`UNION( $x, y$ ) // one root points to the other`

`LINK(FIND( $x$ ), FIND( $y$ ))`

`LINK( $x, y$ ) // implements union by rank`

`if  $x.rank > y.rank$  // rank bounds tree depth`

`$y.p = x$  // connect shallower tree to deeper`

`else  $x.p = y$`

`if  $x.rank == y.rank$  // only equal ranks increases`

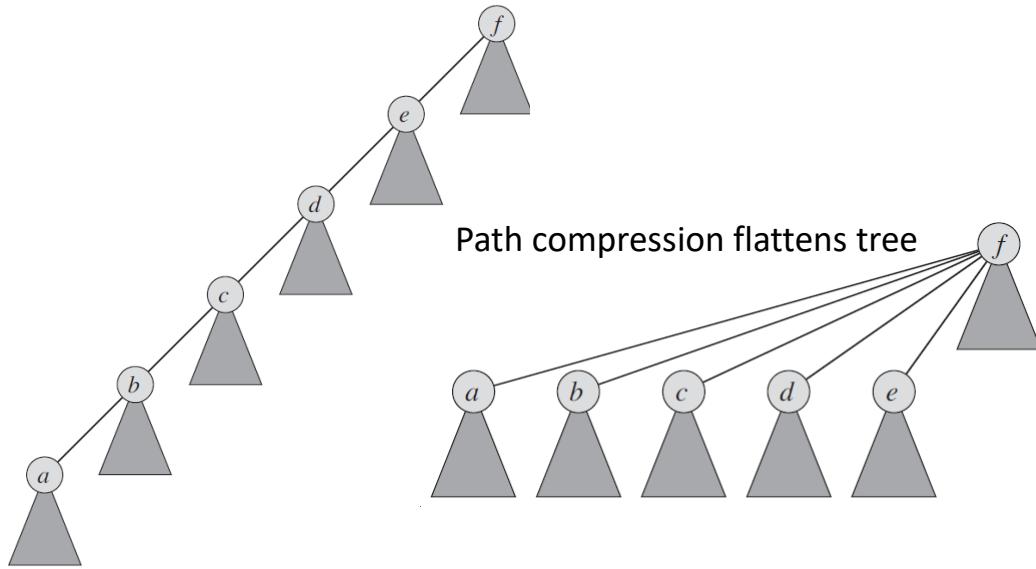
`$y.rank = y.rank + 1$  union's root rank`

September 2024

Algorithms and DS II: Disjoint Sets

12

12



September 2024

Algorithms and DS II: Disjoint Sets

13

13



```

FIND (x) // simultaneous find and path compression
  if x ≠ x.p // check for root
    x.p = FIND(x.p) // recursive upward pass
  return x.p // return the root which points to itself

```

FIND is two-pass method. As it recurs, it makes one pass upwards to find the root.

As the recursion unwinds, it makes a second pass back downwards to update each node to point directly to the root.

September 2024

Algorithms and DS II: Disjoint Sets

14

14



**Lemma 2:** Number of nodes in a tree  $T$  of height  $h$  satisfies  $|T| \geq 2^h$ .

**Proof:** By induction on series of operations.

Base: Let  $x$  and  $y$  be single node trees created by MAKE, where  $x.rank = y.rank = 0$ . Then  $|T| = 1 \geq 2^0$  indeed.

Subsequent MAKEs and FINDs make no change until first UNION.

UNION( $x, y$ ) results in the smallest tree of height 1, yielding  $y.rank = 1$ . Then  $|T| = 2 \geq 2^1$  indeed.

Induction: after  $i$  operations there is  $|T_1| \geq 2^{h_1}$  and  $|T_2| \geq 2^{h_2}$ .



Subsequent MAKEs do not change depth, whereas subsequent FINDs may only flatten  $T_1$  and  $T_2$ , i.e., reducing  $h_1$  and  $h_2$ .

Upon UNION by rank  $T = T_1 \cup T_2$ ,  $|T| = |T_1| + |T_2| \geq 2^{h_1} + 2^{h_2}$ .

If  $h_1 = h_2$ ,  $h(T) = h_1 + 1 \Rightarrow |T| \geq 2^{h_1} + 2^{h_2} = 2^{h_1+1} = 2^{h(T)}$ .

If  $h_1 > h_2$ ,  $h(T) = h_1 \Rightarrow |T| \geq 2^{h_1} + 2^{h_2} > 2^{h_1} = 2^{h(T)}$ . ■

**Corollary:** The height of trees is bounded by  $O(\log n)$ . (**HW**)

**Corollary:** Time complexity of any operation is bounded by  $O(\log n)$ , and a series of  $m$  operation is bounded by  $O(m \log n)$ .



## Analysis of Union by Rank and Path Compression

Root:

- MAKE creates root with rank 0
- UNION may increase root's rank by 1.

Internal node:

- Never turns to be a root
- Its rank never changed
- Path compression may reduce its subtree height
- The rank of its root is non decreasing

Rank is an upper bound of subtree depth

September 2024

Algorithms and DS II: Disjoint Sets

17

17



**Lemma 3:** If  $u$  is the parent of  $v$ , then  $\text{rank}(v) \leq \text{rank}(u)$ , with strict inequality if  $u \neq v$ .

**Proof:** By induction on the series of operations on underlying DS.

Base: First operations  $\text{MAKE}(u)$  and  $\text{MAKE}(v)$ , yields single node DS with rank 0.

Subsequent MAKEs and FINDs make no change until first UNION.

$\text{UNION}(u, v)$  makes  $u$  parent of  $v$ ,  $\text{rank}(v) = 0$  and  $\text{rank}(u) = 1$ .

September 2024

Algorithms and DS II: Disjoint Sets

18

18



Induction:  $\text{rank}(v) \leq \text{rank}(u)$  holds after  $i$ th operation.

The  $(i + 1)$ th operation can be any of:

MAKE does not affect the  $\text{rank}(v) \leq \text{rank}(u)$  in any way.

UNION connects root of smaller rank to higher one, and if equal, the parent's rank is increased by 1.

FIND employs path compression, where descendant is connected to its ancestor root, which by induction its rank is higher. ■



We shall prove that a sequence of  $m$  disjoint-set operations on  $n$  objects takes  $O(m \cdot \alpha(n))$  time, where  $\alpha(n)$  grows very slowly.

$\alpha(n)$  counts how many times log is applied until

$$\log(\log(\dots \log(n) \dots)) \leq 1.$$

Formally,

$$\alpha(n) \triangleq \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + \alpha(\log n) & \text{if } n > 1 \end{cases}$$



For argument  $n = 2^{2^{\dots^2}}$ ,  $\alpha(n)$  is number of 2s in the tower.

$$\alpha(2) = \alpha(2^1) = 1 + \alpha(1) = 1.$$

$$\alpha(4) = \alpha(2^2) = 1 + \alpha(2) = 2.$$

$$\alpha(16) = \alpha(2^4) = \alpha(2^{2^2}) = 1 + \alpha(4) = 3.$$

$$\alpha(65536) = \alpha(2^{16}) = \alpha(2^{2^2}) = 1 + \alpha(16) = 4.$$

$$\alpha(2^{65536}) = \alpha(2^{2^{2^2}}) = 1 + \alpha(65536) = 5.$$



**Lemma 4:** A forest of  $n$  nodes has at most  $\lfloor n/2^r \rfloor$  nodes of rank  $r$ .

**Proof:** Let a root  $x$  change its rank from  $r - 1$  to  $r$  upon UNION, and let  $T_x$  be the implied tree. Mark  $x$ .

Since rank bounds height, from lemma 2 there is  $|T_x| \geq 2^r$ , where rank of all  $T_x$ 's internal nodes is smaller than  $r$ .

The rank of  $T_x$ 's internal nodes never change, and  $x$  rank can only increase.  $\Rightarrow$  Node is marked at most once.

$\Rightarrow$  Markers of rank  $r$  occur at most  $\lfloor n/2^r \rfloor$  times. ■



**Theorem 5:** Total run time of  $m$  operations of which  $n$  are MAKE is  $O(m \cdot \alpha(n))$ .

**Proof:** By amortized analysis. For  $k = \alpha(n)$ , we define  $\alpha^{-1}(k) = n$ .

Divide the nodes into sets  $S_i$  as follows:

$$S_0 = \{x \mid x.rank \in [0, 2^0 - 1]\}, \text{ i.e., } x.rank = 0.$$

$$S_1 = \{x \mid x.rank \in [2^0, 2^1 - 1]\}, \text{ i.e., } x.rank = 1.$$

$$S_2 = \{x \mid x.rank \in [\alpha^{-1}(1), \alpha^{-1}(2) - 1]\}, \text{ i.e., } x.rank = 2, 3.$$

$$S_3 = \{x \mid x.rank \in [\alpha^{-1}(2), \alpha^{-1}(3) - 1]\}, \text{ i.e., } x.rank = 4, 5, \dots, 15.$$

September 2024

Algorithms and DS II: Disjoint Sets

23

23



And in general

$$S_i = \{x \mid x.rank \in [\alpha^{-1}(i - 1), \alpha^{-1}(i) - 1]\}.$$

No rank exceeds  $n \Rightarrow$  total number of sets does not exceed  $\alpha(n)$ .

By lemma 4, the number of nodes which ranks are in  $S_k$  (ranks in  $[\alpha^{-1}(k - 1), \alpha^{-1}(k) - 1]$ ) is bounded by

$$(1) \quad \sum_{j=\alpha^{-1}(k-1)}^{\alpha^{-1}(k)-1} \lfloor n/2^j \rfloor < \frac{n}{2^{\alpha^{-1}(k-1)}} \sum_{j=0}^{\infty} 2^{-j} = \frac{2n}{2^{\alpha^{-1}(k-1)}} = \frac{2n}{\alpha^{-1}(k)}.$$

September 2024

Algorithms and DS II: Disjoint Sets

24

24



The total time consumed by the operations can be associated to the traversed links.

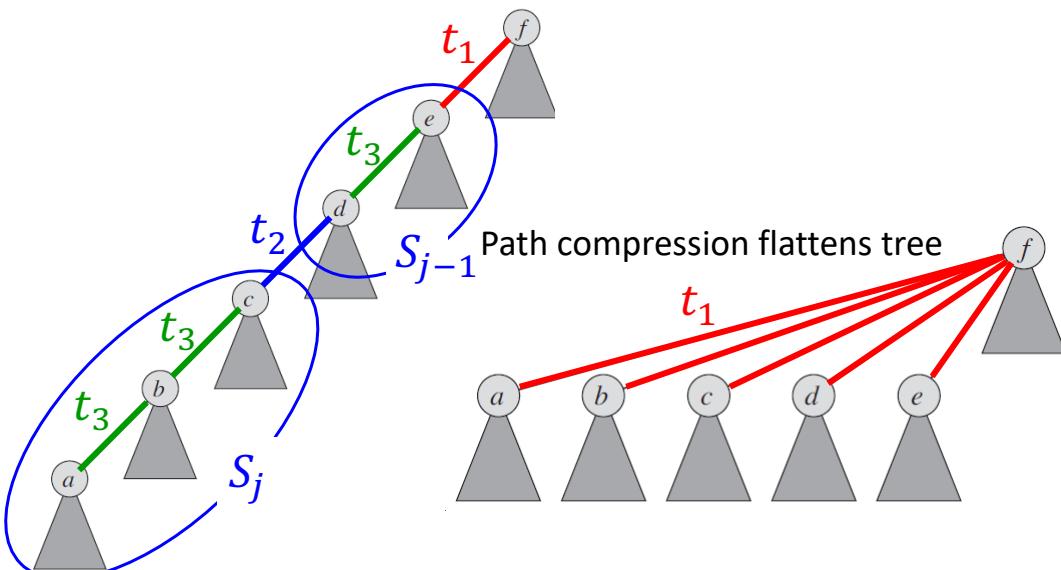
Consider the links traversed in all FINDs among the  $m$  operations. MAKE and UNION anyway contribute totally  $O(n)$  time.

$t_1$  = # links traversed to a root

$t_2$  = # links traversed across the sets  $S_0, S_1, \dots, S_{\alpha(n)}$

$t_3$  = # links traversed within the sets  $S_0, S_1, \dots, S_{\alpha(n)}$

The total complexity of the  $m$  operations is  $O(t_1 + t_2 + t_3)$ .





In path compression of FIND one link is connected directly to root.  
Hence,  $t_1 = O(m)$ .

In the worst case any operation traverses across all  $S_0, S_1, \dots, S_{\alpha(n)}$ ,  
yielding a bound  $t_2 = O(m \cdot \alpha(n))$ .

To account the links within  $S_k$ ,  $1 \leq k \leq \alpha(n)$ , consider the path  
 $v_0 \rightarrow v_1 \rightarrow v_2 \dots \rightarrow v_p$ .

By lemma 3 their rank is strictly increasing, hence the paths length is  
bounded by  $(\alpha^{-1}(k) - 1) - \alpha^{-1}(k - 1) < \alpha^{-1}(k)$ .



Hence

(1)

$$\begin{aligned} t_3 &\leq \sum_{k=0}^{\alpha(n)} \sum_{v \in S_k} \alpha^{-1}(k) = \sum_{k=0}^{\alpha(n)} \alpha^{-1}(k) \frac{2n}{\alpha^{-1}(k)} \\ &= 2n \cdot \alpha(n) = O(n \cdot \alpha(n)). \end{aligned}$$

Summing  $t_1$ ,  $t_2$  and  $t_3$ , there is

$$\begin{aligned} t_1 + t_2 + t_3 &\leq O(m) + O(m \cdot \alpha(n)) + O(n \cdot \alpha(n)) \\ &= O(m \cdot \alpha(n)). \blacksquare \end{aligned}$$

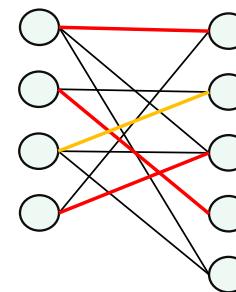
## מבנה נתונים ואלגוריתמים 2

### תרגול: שידור מקסימום וכיסוי בצלמים ממושקל בגרף דו"צ

1

### שידור מקסימום בגרף דו"צ

- ◻ גראף דו-צדדי ( $G = (L \cup R, E)$ ):
  - קבוצת הצלמים של מחולקת לשני תת-קבוצות או צדדים  $L$  ו-  $R$ .
  - הקשתות שלו מחוברות בין צלמים השייכים לצדדים שונים, כלומר  $R \times L \subseteq E$ .



2

- ◻ שידור:
  - תת-קבוצה של קשתות  $M$  שזחوت בצלמים
  - כל צמת נוגע בלבד יוטר מקשחת אחת מ-  $M$ .
- ◻ שידור מקסימום:
  - שידור שגודלו מירבי.

- ◻ שידור מושלם:
  - שידור  $M$  שמקיים  $|R| = |M| = |L|$ .

2

1

### דוגמה

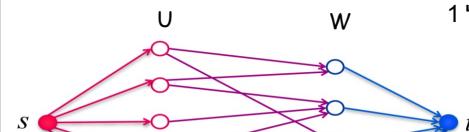
- לצער בעלי חיים הגיעו 100 משפחות במטרה לאמצ  
כלב. בכלבייה יש 100 כלבים לאיום ולכל משפחה  
רשימה של כלבים אותם היא מוכנה לאמצ.
- מצאו אלגוריתם שימצא את מספר האימוצים  
המקסימלי האפשר.



3

### דוגמה

- נציג את המשפחות והכלבים בגרף דו צדי ( $E, W, U = G$ ):
- המשפחות יהיו מצד השמאלי של הגרף  
הכלבים מצד ימני של הגרף
- הקשתות ייצגו את רשימת הכלבים אותה כל משפחה מוכנה לאמצ ויהו  
מכוננות שמאל לימין.
- בהינתן הגרף  $G$  בניית רשת זרימה  $H$  באופן הבא:
- נוסיף קודקוד  $S$  וניתור קשתות ממנו לכל קודקוד  $U$
- נוסיף קודקוד  $t$  וניתור קשתות אליו מכל קודקוד  $W$
- ניתן לכל הקשתות קיבול 1



4

2

## דוגמה

- קיבלנו רשת זרימה, נרץ עליה אלג' למציאת זרימה מקסימלית מ-S ל-t
- כפי שהוכח בהרצאה מתקיים שם  $f$  היא זרימה שלמה ב-H, אז קיים שידור ב-  $G$  כך ש-  $|f| = |M|$ .



5

## בעיית הכיסוי בצלמתים

- יהיו  $(V, E) = G$ , גרף לא מכוון.  $V \subseteq U$  נקרא כיסוי בצלמתים (Vertex Cover) של הגרף אם לכל קשת  $E \in (v, u)$  מתקיים  $v \in U$  או  $u \in U$ .
- במקרה אחר, סילוק  $U$  מהגרף מותיר אותו חסר קשטוות.



6

## הגדרת הבעיה

□ **הבעיה:** בהינתן גרף עם משקלים על הצלמתים מצא  $C$  במשקל מינימלי.

### □ העזרות:

- .1. ניתן להניח שהמשקלים אי-שליליים
- .2. בגרף כללי הבעיה היא NP-קשה

□ נפתרו את בעיית הכיסוי בצלמתים בגרף דו צדדי.

□ נשתמש בזרימה על מנת לפתור את הבעיה.

7

## שלב 1 - בניית רשת N

□ נגדיר את הגרף המכוון  $(G', E', V') = G'$  באופן הבא:

$V' = V \cup \{s, t\}$

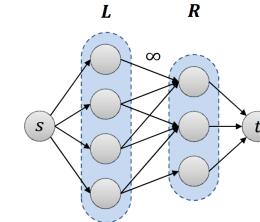
□  $E'$  כולל את הקשתות:

□ לכל  $L \in \mathcal{L}$  נווסף קשת  $s-L-u$  כר ש- $(u)$

□ לכל  $R \in \mathcal{R}$  נווסף קשת  $-L-t$  כר ש- $(t)$

□ נכון כל קשת  $(u, v) \in E$  מוש- $L$ - $v$  ונגדיר ש- $\infty = (v, u)$

$E' = \{su \mid u \in L\} \cup \{vt \mid v \in R\} \cup \{uv \mid uv \in E, u \in L, v \in R\}$



8

## טענה 1

□ **כיסוי בצמתים A** בגראף  $G$  מגדיר חתך  $(\bar{S}, S)$  שקיובלו ( $A$ ) ו.

הוכחה:

□ בהינתן כיסוי בצמתים  $A$  גדר חתך  $(S, \bar{S})$  ע"י:

$$S = (L \setminus A) \cup (R \cap A)$$

ו  $\{s\} \cup \{t\}$  נקבע בחתך  $(S, \bar{S})$ , קשותות החתך כוללת:

1. קשותות מ- $s$  ל- $L \setminus A$

2. קשותות מ- $R \cap A$  ל- $t$

לא תמכננה קשותות מ- $A \setminus L$  ל- $R \setminus A$  כי אחרת  $A$  אינו כיסוי.

לכן

$$\begin{aligned} C(S, \bar{S}) &= \sum_{u \in A \cap L} c(s, u) + \sum_{v \in A \cap R} c(v, t) = \\ &= \sum_{u \in A \cap L} w(u) + \sum_{v \in A \cap R} w(v) = w(A) \end{aligned}$$

9

## טענה 2

□ כל חתך  $(S, \bar{S})$  בעל קיבול סופי מגדיר כיסוי בצמתים  $A$  במשקל  $C(S, \bar{S})$ .

הוכחה:

□ בהינתן חתך  $(S, \bar{S})$  גדר כיסוי  $A$  ע"י:

$$A = (S \cap L) \cup (S \cap R)$$

□ כיוון שהחתך בעל קיבול סופי הוא אינו מכיל קשותות מ- $L$  ל- $R$  שכן אין קשת מ- $S \cap L$  ל- $S \cap R$ .

□ לכך  $A$  מהו כיסוי בצמתים.

□ כמו כן

$$\begin{aligned} w(A) &= w(A \cap L) + w(A \cap R) = \sum_{u \in A \cap L} w(u) + \\ &+ \sum_{v \in A \cap R} w(v) = \sum_{u \in A \cap L} c(s, u) + \sum_{v \in A \cap R} c(v, t) = C(S, \bar{S}) \end{aligned}$$

10

## האלגוריתם למציאת VC מינימום

- קלט: גרף דו צדדי ( $G = (L \cup R, E)$ ).
  - פלט: כיסוי בצמתים  $A$  של הגרף  $G$ .
- .1. בונה את רשת הזרימה  $N$  כפי שהוגדר בשקפים הקודמים.
  - .2. הרץ את האלגוריתם Ford-Fulkerson לחישוב זרימת מקסימום ב- $N$ .
  - .3. מצא חתך מינימום (לפי משפט ה- Max-Flow Min-Cut )
  - .4. החזר את הקבוצה:  $A = (S \cap L) \cup (\bar{S} \cap R)$

11

## מסקנה

- משקל VC מינימום = קיבול חתך מינימום
  - לכן כדי לפתור את הבעיה נמצא זרימה מקסימלית ברשת ומתוכה נמצא חתך מינימום, וממנו נחלץ את ה-VC המתאים.
- הערות:
- במקרים להשתמש ב- $\infty$  לקיבול קשת  $(v,u)$  מ- $L$  ל- $R$  נכון להשתמש בערך  $+1$  ב- $(v,u)$ .
  - קיבול זה מבטיח שהקשת לא תהיה רויה לעולם, ולכן קבוצת הזרימות החוקיות לא תשתנה.
  - בבעיה הלא ממושקלת הרשת זהה לרשת שבנוינו עבור בעיית השידור מקסימום ולכן נקבל זמן ריצה של  $\Theta(m \cdot \sqrt{n})$

12

## מבנה נתונים ואלגוריתמים 2 תרגול: Fast matrix multiplication

Based on: Introduction to Algorithms  
by Cormen, Leiserson, Rivest, and Stein

13

13

### הכפלת מטריצות בסיסית

- נרצה להכפיל שני מטריצות מוגדל  $2 \times 2$  לדוגמה

$$: C = A \times B$$

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} & c_{21} &= a_{21}b_{11} + a_{22}b_{21} \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} & c_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

- ניתן לבצע הכפלת מטריצות מוגדל  $2 \times 2$  עם 8 פעולות כפל.

14

14

## הכפלת מטריצות בסיסית

### Algorithm

```
void matrix_mult (){
    for (i = 1; i <= N; i++) {
        for (j = 1; j <= N; j++) {
            compute Cij;
        }
    }
}
```

### Time analysis

$$C_{i,j} = \sum_{k=1}^N a_{i,k} b_{k,j}$$

$$\text{Thus } T(N) = \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N c = cN^3 = O(N^3)$$

15

## אלגוריתם הפרד-ומשל פשטן להכפלת מטריצות

- נשתמש באלגוריתם הפרד-ומשל לחשב את תוצאת המטריצה  $C = A \cdot B$

- יהו  $A$  ו  $B$  שני מטריצות מוגדל אח, כאשר  $\ell$  חזקה של 2

- נחלק את  $A, B$  ו  $C$  לארבע מטריצות מוגדל  $n/2 \times n/2$

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

- בכל אחד מארבעת המכפלות הנ"ל מופיעים שני מכפלות של מטריצות מוגדל  $n/2 \times n/2$  וחיבור של התוצאות מוגדל  $n/2 \times n/2$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} \quad c_{21} = a_{21}b_{11} + a_{22}b_{21}$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22} \quad c_{22} = a_{21}b_{12} + a_{22}b_{22}$$

16

15

16

## אלגוריתם הפרד-ומשול פשוט להכפלת מטריצות

SQUARE-MATRIX-MULTIPLY-RECURSIVE( $A, B$ )

```

1    $n = A.\text{rows}$ 
2   let  $C$  be a new  $n \times n$  matrix
3   if  $n == 1$ 
4      $c_{11} = a_{11} \cdot b_{11}$ 
5   else partition  $A, B$ , and  $C$  as in equations (4.9)
6      $C_{11} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{11})$ 
        +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{21})$ 
7      $C_{12} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{12})$ 
        +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{22})$ 
8      $C_{21} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{11})$ 
        +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{21})$ 
9      $C_{22} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{12})$ 
        +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{22})$ 
10  return  $C$ 
```

17

17

## שיטת האב (המאסטר) - תזכורת

Let  $a \geq 1$  and  $b > 1$  be constants, let  $f(n)$  be a function, and let  $T(n)$  be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret  $n/b$  to mean either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ . Then  $T(n)$  has the following asymptotic bounds:

1. If  $f(n) = O(n^{\log_b a - \epsilon})$  for some constant  $\epsilon > 0$ , then  $T(n) = \Theta(n^{\log_b a})$ .
2. If  $f(n) = \Theta(n^{\log_b a})$ , then  $T(n) = \Theta(n^{\log_b a} \lg n)$ .
3. If  $f(n) = \Omega(n^{\log_b a + \epsilon})$  for some constant  $\epsilon > 0$ , and if  $af(n/b) \leq cf(n)$  for some constant  $c < 1$  and all sufficiently large  $n$ , then  $T(n) = \Theta(f(n))$ . ■

18

9

## ניתוח סיבוכיות

- נכתוב את משוואת הרקורסיה להכפלת מטריצות:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

- לפי שיטת המאסטר :

$$a=8, b=2, f(n) \in O(n^2) \quad \circ$$

$$\log_2 8 = 3 \rightarrow f(n) \in O(n^3) \rightarrow T(n) \in O(n^3) \quad \circ$$

19

## Strassens's Matrix Multiplication

- בשנת 1969 סטרנס הראה כי ניתן לבצע כפל מטריצות בגודל  $2 \times 2$  באמצעות 7 פעולות כפל + 18 פעולות חיבור או חיסור.

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix} = \begin{pmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{pmatrix}$$

$$m_1 = (a_{00}+a_{11}) * (b_{00}+b_{11})$$

$$m_2 = (a_{10}+a_{11}) * b_{00}$$

$$m_3 = a_{00} * (b_{01}-b_{11})$$

$$m_4 = a_{11} * (b_{10}-b_{00})$$

$$m_5 = (a_{00}+a_{01}) * b_{11}$$

$$m_6 = (a_{10}-a_{00}) * (b_{00}+b_{01})$$

$$m_7 = (a_{01}-a_{11}) * (b_{10}+b_{11})$$

20

19

20

## Strassens's algorithm

- .1. מטריצות הקלט  $A$  ו  $B$  וכן מטריצת התוצאה  $C$  יחולקו לחתמי מטריצות בגודל  $n/2 \times n/2$

- סיבוכיות זמן:  $O(1)$  לחישובי אינדקסים

$$\begin{pmatrix} A_{11} & | & A_{12} \\ \hline A_{21} & | & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & | & B_{12} \\ \hline B_{21} & | & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & | & C_{12} \\ \hline C_{21} & | & C_{22} \end{pmatrix}$$

Each block is  $(n/2) \times (n/2)$

21

## Strassens's algorithm

- .2. צריכים 10 מטריצות  $S_1, S_2, \dots, S_{10}$  בגודל  $n/2 \times n/2$ , כאשר כל אחת מהן היא סכום או חיסור של שניים מהמטריצות שוחשבו בשלב 1.

- סיבוכיות זמן:  $O(n^2)$  לחישוב כל 10 המטריצות
- $$\begin{aligned} S_1 &= B_{12} - B_{22}, \\ S_2 &= A_{11} + A_{12}, \\ S_3 &= A_{21} + A_{22}, \\ S_4 &= B_{21} - B_{11}, \\ S_5 &= A_{11} + A_{22}, \\ S_6 &= B_{11} + B_{22}, \\ S_7 &= A_{12} - A_{22}, \\ S_8 &= B_{21} + B_{22}, \\ S_9 &= A_{11} - A_{21}, \\ S_{10} &= B_{11} + B_{12}. \end{aligned}$$

22

## Strassens's algorithm

.**3.** ע"י שימוש בתתי המטריצות שלב 1, ו 10 המטריצות שחושבו בשלב 2, באופן רקורסיבי יחושו 7 מכפלות של מטריצות לתוך  $P_1, P_2, \dots, P_7$ .

כל מטריצה תוצאה  $P_i$  היא מוגדל  $n/2 \times n/2$

$$\begin{aligned}P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}, \\P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}, \\P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}, \\P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}, \\P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}, \\P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}, \\P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}.\end{aligned}$$

23

23

## Strassens's algorithm

.**4.** תת-מטריצות  $C_{11}, C_{12}, C_{21}, C_{22}$  של מטריצת התוצאה  $C$  יוחשבו ע"י קומבינציות שונות של פעולות חיבור וחיסור של מטריצות  $P_i$ .

סיבוכיות זמן:  $O(n^2)$  לחישוב ארבעת תת-מטריצות

$$C_{11} = P_5 + P_4 - P_2 + P_6.$$

$$C_{12} = P_1 + P_2,$$

$$C_{21} = P_3 + P_4$$

$$C_{22} = P_5 + P_1 - P_3 - P_7,$$

24

24

## Strassen Algorithm

```
Strassen(A, B)
if A.length == 1 then
    return A[1] · B[1]
end if
Let C be a new n by n matrix
A11 = A[1..n/2][1..n/2]
A12 = A[1..n/2][n/2 + 1..n]
A21 = A[n/2 + 1..n][1..n/2]
A22 = A[n/2 + 1..n][n/2 + 1..n]
B11 = B[1..n/2][1..n/2]
B12 = B[1..n/2][n/2 + 1..n]
B21 = B[n/2 + 1..n][1..n/2]
B22 = B[n/2 + 1..n][n/2 + 1..n]
S1 = B12 - B22
S2 = A11 + A12
S3 = A21 + A22
S4 = B21 - B11
S5 = A11 + A22
```

25

## Strassen Algorithm

```
S6 = B11 + B22
S7 = A12 - A22
S8 = B21 + B22
S9 = A11 - A21
S10 = B11 + B12
P1 = Strassen(A11, S1)
P2 = Strassen(S2, B22)
P3 = Strassen(S3, B11)
P4 = Strassen(A22, S4)
P5 = Strassen(S5, S6)
P6 = Strassen(S7, S8)
P7 = Strassen(S9, S10)
C[1..n/2][1..n/2] = P5 + P4 - P2 + P6
C[1..n/2][n/2 + 1..n] = P1 + P2
C[n/2 + 1..n][1..n/2] = P3 + P4
C[n/2 + 1..n][n/2 + 1..n] = P5 + P1 - P3 - P7
return C
```

26

25

26

## ניתוח סיבוכיות

- נכתב את משווהת הרקורסיה:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

- לפי שיטת המאוסטר:

- $a=7, b=2, f(n) \in O(n^2)$

- $\log_2 7 = 2.71 \rightarrow f(n) \in O(n^{2.71}) \rightarrow T(n) \in O(n^{2.71})$

27

## דוגמה 1

$$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$$

השתמשו באלגוריתם סטרון להכפלת המטריצות

$$\begin{array}{llll} S_1 = 8 - 2 = 6 & S_6 = 6 + 2 = 8 & P_1 = 6 & P_5 = 48 \\ S_2 = 1 + 3 = 4 & S_7 = 3 - 5 = -2 & P_2 = 8 & P_6 = -12 \\ S_3 = 7 + 5 = 12 & S_8 = 4 + 2 = 6 & P_3 = 72 & P_7 = -84 \\ S_4 = 4 - 6 = -2 & S_9 = 1 - 7 = -6 & P_4 = -10 & \\ S_5 = 1 + 5 = 6 & S_{10} = 6 + 8 = 14 & & \end{array}$$

נקבל את מטריצת התוצאות

$$\begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}$$

28

27

28

## דוגמה 2

- כיצד הייתם מושנים את אלגוריתם סטרון להכפלת מטריצות מגודל  $A \times B$  עבור  $A$  שאינה חזקה של 2?
- הראו שסיבוכיות האלגוריתם היא  $\Theta(n^{\lg 7})$ .

29

29

## דוגמה 2

- נרףד באפסים את מטריצות הקלט למטריצה בגודל חזקה שלמה של 2.
- הריפוד למספר הבא שהוא חזקה שלמה של 2 (נקרא לגודל זה 3) לכל היותר יכפיל את הערך  $A$
- נקבל:
$$m^{\lg 7} \leq (2n)^{\lg 7} = 7n^{\lg 7} \in O(n^{\lg 7})$$
- וגם:
$$m^{\lg 7} \geq n^{\lg 7} \in \Omega(n^{\lg 7})$$
- משני אי-השוויונים נקבל:  $\Theta(n^{\lg 7})$

30

15

# מבנה נתונים ו알גוריתמים 2

## תירגול: Skip List

Based on: Introduction to Algorithms  
by Cormen, Leiserson, Rivest, and Stein

1

1

### דוגמה 1

- מכנים איברים לטור רשימת נסמן ב- $K$  את המשתנה המקרי (האי-שלילי) המציין את גובה או מספר הרמות בראשימת הדילוגים.
- נגיד המשתנה מקרי:  $n_2 = \max(0, K - \log_2 n)$
- הראו כי התוחלת של המשתנה המקרי היא  $O(1)$
- השתמשו בסעיף קודם כדי להראות כי  $E[K] \leq \log_2 n + O(1)$

heaps

2

2

1

## פתרון סעיף א

$$\begin{aligned} E[X] &= \sum_{i \geq 1} \Pr[X \geq i] = \sum_{i \geq 1} \Pr[K \geq \log_2 n + i] = \\ &\sum_{i \geq 1} \Pr[j \text{ exists s.t } x_j \text{ reaches layer } \log_2 n + i] \leq \\ &\sum_{j=1}^n \sum_{i \geq 1} \Pr[x_j \text{ reaches layer } \log_2 n + i] \leq \\ &\sum_{j=1}^n \sum_{i \geq 1} 2^{-\log_2 n - i} = \sum_{i \geq 1} n \frac{1}{n * 2^i} = 1 \end{aligned}$$

heaps 3

3

## פתרון סעיף ב

- שימוש ישיר בתכונות התוחלת נוטן כי:
- $$E[K] = E[\log_2 n + (K - \log_2 n)] \leq E[X]$$
- וע"י הצבה של התוצאה מהסעיף הקודם קיבל את המבוקש.

heaps 4

4

2

# מבנה נתונים ו알גוריתמים 2

## Treaps: תרגול

Based on: Introduction to Algorithms  
by Cormen, Leiserson, Rivest, and Stein

5

5

### דוגמה 1

- פועלת TREAP-INSERT מבצעת חישוב ולאחר מכן רצף של סיבובים. למרות שיחסוש וסיבובים הם בעלי זמן ריצה אסימפטוטי זהה, יש להם עליות שונות בפועל. חישוב קורא מידע מה Treap מבלי לשנות אותו, בעוד שסיבוב משנה את מצביעי ההורה והיל' בתוך ה Treap ברוח המחשבים, פעולות קריאה הן מהירות בהרבה מפעולות כתיבה. לכן נרצה ש TREAP-INSERT יבצע כמה שפחות סיבובים. נראה כי מספר הסיבובים הצפוי מוגבל בקבוע.
- כדי להראות תכונה זו, נזדקק לכמה הגדירות. עמוד השדרה השמאלי של עץ חישוב בינארי T הוא הנטייב שמתחל בשורש ומגיע לפרק עם המפתח הקטן ביותר. במקרים אחרים, עמוד השדרה השמאלי הוא הנטייב המקסימלי מהשורש שמורכב רק מקצוות שמאליים. באופן סימטרי, עמוד השדרה ימני של T הוא הנטייב המינימלי מהשורש שמורכב רק מקצוות ימניים. אורך עמוד השדרה הוא מספר הצלמתים שהוא מיל.

heaps

6

6

3

## דוגמה 1

- נתבונן ב-  $T$  מיד לאחר שהוכנס  $x$  באמצעות הפעולה  $\text{TREAP-INSERT}$ . נניח ש-  $C$  הוא אורך עמוד השדרה הימני של תת-העץ השמאלי של  $x$ . ו-  $D$  הוא אורך עמוד השדרה השמאלי של תת-העץ הימני של  $x$ . הוכח שמספר הסיבובים הכלל שבוצע במהלך הוכנסה של  $x$  שווה ל- $C+D$ .

heaps

7

7

## פתרון

- נוכיח את הטענה באינדוקציה:
- מקרה בסיס: עבור  $\text{TREAP}$  בעל קודקוד  $z$  נוכיח את הקודקוד  $x$  ונג Lager. כעת גלגלו פעם אחת וגובה תת-העץ הימני או השמאלי הוא 1 ולכן זהו סכום עמודי השדרה כמספר הgalgoles.

heaps

8

8

## פתרון

- צעד האינדוקציה: נניח כי הטענה נכונה עבור  $k$  גלגולים ונוכיח כי היא נכונה עבור  $k+1$ .
- אנו יודעים שלפי הנחת האינדוקציה  $C = D + k$ . נניח כי  $x$  הוא הבן השמאלי של  $z$ . לאחר ביצוע גלגול ימני על  $x$  הופך להיות הבן הימני של  $x$  והבן הימני הקודם של  $x$  הופך להיות הבן השמאלי של  $z$ . لكن עמוד השדרה הימני של  $x$  גדל ב-1. מצד שני עמוד השדרה השמאלי של  $x$  לא השתנה כי הגלגול לא השפיע על תות העץ השמאלי של  $x$ . בסה"כ הראנו כי  $C+D = k+1$  וסיימנו.
- הערכה: המקרה השני סימטרי

heaps

9

9

## מבנה נתונים ואלגוריתמים 2 תירגול: Fast Fourier Transform

Based on: Introduction to Algorithms  
by Cormen, Leiserson, Rivest, and Stein

10

10

5

## דוגמה 1

- יובל המבולבל ניסה לחשב את המכפלה של שני הפולינומים  $x = A(x)$  ו-  $x = B(x)$  ע"י שימוש באלגוריתם ה-FFT:
- הוא קבוע  $a = (0,1)$  ו-  $b = (1,-1)$  וחישב  $\mathcal{F}(a) = \mathcal{F}(b) = (1,-1)$ .
  - הוא חישב את המכפל  $(1,1) \cdot \mathcal{F}(a) \cdot \mathcal{F}(b) = (1,1)$ .
  - הוא חישב  $\mathcal{F}^{-1}(1,1) = (1,0)$ , כלומר הוא קיבל את הפולינום  $1 = C(x)$ , וזה תשובה נכונה.

- א. הסבירו מה היא הטעות שעשאה יובל המבולבל, תחת ההנחה שהוא חישב את  $\mathcal{F}$  ואת  $\mathcal{F}^{-1}$  נכון.  
 ב. בצעו את החישוב הנכון, כולל הכפיל את שני הפולינומים ע"י שימוש ב-FFT.

heaps 11

11

## תזכורת

- מתקנים  $w_n^k = e^{i \frac{2\pi k}{n}}$
- התמרת פוריה הדיסקרטית של  $(a_0, \dots, a_{n-1})$  היא סדרת המספרים  $y_{n-1}, y_{n-2}, \dots, y_0$ , כאשר מתקיים  $y_k = \sum_{j=0}^{n-1} a_j (w_n^k)^j$ .
- התמרת פוריה הדיסקרטית ההפוכה של  $y_0, \dots, y_{n-1}$  היא סדרת המקדים  $(a_0, \dots, a_{n-1})$ , כאשר מתקיים  $a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k (w_n^{-j})^k$ .

heaps 12

12

## פתרונות

□ א. הטעות היא שהיה צריך להאריך את וקטורי המוקדמים לאורך של  $1 - A_2$  כלומר לאורך 3, ולרפק לחזקה של 2, כלומר ל- 4, ולא להתמיר אותם כמו שהם.

□ ב. החישוב הנכון הוא:

$$y^a = (1, i, -1, -i) \quad \textcircled{O}$$

$$y^b = (1, i, -1, -i) \quad \textcircled{O}$$

$$y^c = (1, -1, 1, -1) \quad \textcircled{O}$$

$$c = \frac{1}{4} \cdot (0, 0, 4, 0) = (0, 0, 1, 0) \quad \textcircled{O}$$

heaps 13

13

## דוגמה 2

□ הראו איך להכפיל שני פולינומים  $p + b, cx + ax$  ע"י שימוש ב-3 הכפלות.

□ פתרון:

כל מוקדם הינו הכפלה כלומר  $- (ad, bc, bd, ac)$

heaps 14

14

## דוגמה 2

□ תארו אלגוריתם הפרד ומשול שמכפיל שני פולינומים בעלי  $\chi$  מקדים שעובד בזמן  $(n^{\log_2 3})$

□ פתרון: נכתב כל פולינום בתור-

$$p(x) = p_{even}(x^2) + xp_{odd}(x^2)$$

לכן נשתמש ב3 המכפלות מהסעיף הקודם ונקבל כי אנו יכולים לבצע בצורה זו אלגוריתם הפרד ומשול.

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

לפי נוסחת המאסטר מקבל זמן ריצה של  $O(n^{\log_2 3})$

□ שיעורי בית: הוכיחו נכונות

heaps

15



ChatGPT 4o ▾



Since we zero-padded to length 4, we're working with a vector length  $n = 4$ , and we'll use the 4th roots of unity in the inverse direction to perform the inverse FFT.

The inverse FFT formula for a sequence  $C' = [C'(0), C'(1), C'(2), C'(3)]$  is:

$$C(k) = \frac{1}{n} \sum_{j=0}^{n-1} C'(j) \cdot \omega_n^{-jk}$$

where:

- $n = 4$ ,
- $\omega_n^{-1}$  is the inverse root of unity, specifically  $e^{-2\pi i/4} = i$ ,
- We need to divide by  $n = 4$  at the end to normalize.

The 4th roots of unity (for  $n = 4$ ) in the inverse FFT are:

- $\omega_0^{-1} = 1$
- $\omega_1^{-1} = -i$
- $\omega_2^{-1} = -1$
- $\omega_3^{-1} = i$

Let's calculate each  $C(k)$  for  $k = 0, 1, 2, 3$ .

## Calculation of Each $C(k)$

### Step 1: Calculate $C(0)$

$$C(0) = \frac{1}{4} (C'(0) \cdot \omega_0^{-1} + C'(1) \cdot \omega_0^{-1} + C'(2) \cdot \omega_0^{-1} + C'(3) \cdot \omega_0^{-1})$$

Since  $\omega_0^{-1} = 1$ , this becomes:

$$C(0) = \frac{1}{4} (6 + (1 + 3i) + 0 + (1 - 3i)) = \frac{1}{4} (6 + 1 + 3i + 1 - 3i)$$

$$C(0) \downarrow \frac{1}{4} \cdot 8 = 2$$

Message ChatGPT



ChatGPT can make mistakes. Check important info.























ChatGPT 4o ▾



- $C'(\omega_1) = A'(\omega_1) \cdot B'(\omega_1) = (2 + i) \cdot (1 + i) = 1 + 3i$
- $C'(\omega_2) = A'(\omega_2) \cdot B'(\omega_2) = 1 \cdot 0 = 0$
- $C'(\omega_3) = A'(\omega_3) \cdot B'(\omega_3) = (2 - i) \cdot (1 - i) = 1 - 3i$

Thus,  $C' = [6, 1 + 3i, 0, 1 - 3i]$ .

6. **Inverse FFT (IFFT) to Interpolate Back:** Apply the inverse FFT to transform  $C'$  back to the time domain, dividing by the length (4) to obtain the final coefficients.

After interpolation, we find:

$$C(x) = 2 + 3x + x^2$$

Thus, the coefficient of  $x^2$ , or  $C(2)$ , is indeed 1.

show me how you now calculate the inverse FFT in detail



Let's go through the **Inverse FFT (IFFT)** calculation in detail, starting from the pointwise multiplication results. The goal of the inverse FFT is to take the frequency-domain representation we obtained (after pointwise multiplication) and return it to the time-domain, or coefficient form, of the polynomial  $C(x)$ .

## Setup for Inverse FFT

After the pointwise multiplication, we have:

$$C' = [6, 1 + 3i, 0, 1 - 3i]$$

Since we zero-padded to length 4, we're working with a vector length  $n = 4$ , and we'll use the 4th roots of unity in the inverse direction to perform the inverse FFT.

The inverse FFT formula for a sequence  $\downarrow = [C'(0), C'(1), C'(2), C'(3)]$  is:

Message ChatGPT



ChatGPT can make mistakes. Check important info.























$$A(x) = 2 + x \quad B(x) = 1 + x$$

$$n=4 \rightarrow \text{FFT}(2, 1)$$

$$\omega_n^0 = 1 \quad \omega_n^1 = i \quad \omega_n^2 = -1 \quad \omega_n^3 = -i$$

$$\text{FFT}((2, 1))$$

$$w \leftarrow 1$$

$$a \leftarrow (2)$$

$$a' \leftarrow (1)$$

$$y^0 \leftarrow \text{FFT}((2)) \leftarrow 2$$

$$y^1 \leftarrow \text{FFT}((1)) \leftarrow 1$$

$$y_0 = 2 + (1 \cdot 1) = 3, \quad A''(x) = A(x) = A^0(x) + x A^1(x)$$

$$y_3 = 2 - 1 \cdot 1 = 1$$

$$y_1 = 2 + i \cdot 1 = 2 + i$$

$$y_4 = 2 - i \cdot 1 = 2 - i$$

$$2 - 2i - i + (-1)$$

$$\text{FFT}((1, 1))$$

$$a^0 \leftarrow 1$$

$$a' \leftarrow 1$$

$$y^0 \leftarrow \text{FFT}(1)$$

$$y^1 \leftarrow \text{FFT}(1)$$

$$\{A(0) = A^0(0) + x A^1(0), A^0(0) = 1, A^1(0) = 1\}$$

$$y_0 = 1 + 1 \cdot 1 = 2$$

$$y_3 = 1 - 1 \cdot 1 = 0$$

$$y_1 = 1 + i \cdot 1 = 1 + i$$

$$y_4 = 1 - i \cdot 1 = 1 - i$$

$$\begin{array}{c|ccccc} & A(x) & (w^0, 3)(w^1, 2+i)(w^2, 1) & (w^3, -i) \\ & B(x) & (w^0, 2)(w^1, 1+i)(w^2, 0) & (w^3, 1-i) \\ & C(x) & (w^0, 6)(w^1, 1+3i)(w^2, 0) & (w^3, 1-3i) \end{array}$$

$$C^0 = \frac{1}{4} [6 + (1+3i) + 0 + (1-3i)] = 2$$

$$C^1 = \frac{1}{4} [6 \cdot 1 + (1+3i) \cdot (-i) + 0 + (1-3i)i] = \frac{1}{4} [6 - i + 3 + i + 3] = \frac{12}{4} = 3$$

$$C^2 = \frac{1}{4} [6 + (1+3i) \cdot w_n^2 + 0 + (1-3i) w_n^6] = \frac{1}{4} [6 - 1 - 3i + 0 - 1 + 3i] = 1$$

$$C = (2, 3, 1) \rightarrow C(x) = 2 + 3x + x^2$$

$$\text{FFT}((2))$$

return 2

$$\text{FFT}((1))$$

return 1

$$P(x) = \sum_{j=0}^{n-1} c_j x^j$$

$$g(x) = \sum_{i=0}^{n-1} c_i x^i$$

$$\omega = e^{\frac{2\pi i}{n}} = [w^0, w^1, \dots, w^{n-1}]$$

Split  $P$  into  $P_e, P_o$  and  $g$  into  $g_e, g_o$

$$P(x) = P_e(x^2) + x P_o(x^2)$$

$$P(-x) = P_e(x^2) - x P_o(x^2)$$

?The Fast Fourier Transform (FFT): Most Ingenious Algorithm

```

def FFT(P):
    # P - [p0, p1, ..., pn-1] coeff representation
    n = len(P) # n is a power of 2
    if n == 1:
        return P
    omega = e^(2πi/n)
    Pe, Po = [p0, p1, ..., pn-2], [p1, p3, ..., pn-1]
    ye, yo = FFT(Pe), FFT(Po)
    y = [0] * n
    for j in range(n/2):
        y[j] = ye[j] + omega^j * yo[j]
        y[j + n/2] = ye[j] - omega^j * yo[j]
    return y

```

FFT  $P(x) : [p_0, p_1, \dots, p_{n-1}]$   
 $\omega = e^{\frac{2\pi i}{n}} : [\omega^0, \omega^1, \dots, \omega^{n-1}]$

FFT  $n = 1 \Rightarrow P(1)$

FFT  $P_e(x^2) : [p_0, p_2, \dots, p_{n-2}]$   
 $[\omega^0, \omega^2, \dots, \omega^{n-2}]$   
 $y_e = [P_e(\omega^0), P_e(\omega^2), \dots, P_e(\omega^{n-2})]$

FFT  $P_o(x^2) : [p_1, p_3, \dots, p_{n-1}]$   
 $[\omega^0, \omega^2, \dots, \omega^{n-2}]$   
 $y_o = [P_o(\omega^0), P_o(\omega^2), \dots, P_o(\omega^{n-2})]$

$P(\omega^j) = y_e[j] + \omega^j y_o[j]$   
 $P(\omega^{j+n/2}) = y_e[j] - \omega^j y_o[j]$   
 $j \in \{0, 1, \dots, (n/2 - 1)\}$

$y = [P(\omega^0), P(\omega^1), \dots, P(\omega^{n-1})]$

$$x_k = \frac{1}{n} \sum_{j=0}^{n-1} X_j w_n^{-jk}$$

- $x_k$  are the coefficients we want to recover,
- $X_j$  are the DFT values (the pointwise multiplication result),
- $\omega_n = e^{-2\pi i/n}$  is the primitive nth root of unity (inverse of  $\omega$  used in FFT),
- $n$  is the number of points (in our case, 4).

Let's go through the **Inverse FFT (IFFT)** calculation in detail, starting from the pointwise multiplication results. The goal of the inverse FFT is to take the frequency-domain representation we obtained (after pointwise multiplication) and return it to the time-domain, or coefficient form, of the polynomial  $C(x)$ .

## Setup for Inverse FFT

After the pointwise multiplication, we have:

$$C' = [6, 1 + 3i, 0, 1 - 3i]$$

Since we zero-padded to length  $n = 4$ , we're working with a vector length  $n = 4$ , and we'll use the 4th roots of unity in the inverse direction to perform the inverse FFT.

The inverse FFT formula for a sequence  $C' = [C'(0), C'(1), C'(2), C'(3)]$  is:

$$C(k) = \frac{1}{n} \sum_{j=0}^{n-1} C'(j) \cdot \omega_n^{-jk}$$

### Step 2: Calculate $C(1)$

$$C(1) = \frac{1}{4} (C'(0) \cdot \omega_0^{-1} + C'(1) \cdot \omega_1^{-1} + C'(2) \cdot \omega_2^{-1} + C'(3) \cdot \omega_3^{-1})$$

Substitute the roots of unity values:

$$C(1) = \frac{1}{4} (6 + (-i)(1 + 3i) + (-1) \cdot 0 + i(1 - 3i))$$

Calculating each term:

1.  $(-i)(1 + 3i) = -i + 3 = 3 - i$
2.  $i(1 - 3i) = i - 3 = -3 + i$

So,

$$C(1) = \frac{1}{4} (6 + (3 - i) + 0 + (-3 + i))$$

Simplify:

$$C(1) = \frac{1}{4} \cdot 12 = 3$$

### Step 4: Calculate $C(3)$

$$C(3) = \frac{1}{4} (C'(0) \cdot \omega_0^{-3} + C'(1) \cdot \omega_1^{-3} + C'(2) \cdot \omega_2^{-3} + C'(3) \cdot \omega_3^{-3})$$

Substitute:

$$C(3) = \frac{1}{4} (6 + i(1 + 3i) + (-1) \cdot 0 - i(1 - 3i))$$

Calculating:

1.  $i(1 + 3i) = i + 3 = 3 + i$
2.  $-i(1 - 3i) = -i + 3 = 3 - i$

Then,

$$C(3) = \frac{1}{4} (6 + (3 + i) + 0 + (3 - i)) = \frac{1}{4} \cdot 12 = 3$$

## Final Result

Thus, the coefficients of the polynomial  $C(x) = 2 + 3x + x^2$  are:

$$[2, 3, 1]$$

where:

- $C(0) = 2$  (constant term),
- $C(1) = 3$  (coefficient of  $x$ ),
- $C(2) = 1$  (coefficient of  $x^2$ ).

where:

- $n = 4$ ,
- $\omega_n^{-1}$  is the inverse root of unity, specifically  $e^{-2\pi i/4} = i$ ,
- We need to divide by  $n = 4$  at the end to normalize.

The 4th roots of unity (for  $n = 4$ ) in the inverse FFT are:

- $\omega_0^{-1} = 1$
- $\omega_1^{-1} = -i$
- $\omega_2^{-1} = -1$
- $\omega_3^{-1} = i$

Let's calculate each  $C(k)$  for  $k = 0, 1, 2, 3$ .

### Calculation of Each $C(k)$

#### Step 1: Calculate $C(0)$

$$C(0) = \frac{1}{4} (C'(0) \cdot \omega_0^{-1} + C'(1) \cdot \omega_0^{-1} + C'(2) \cdot \omega_0^{-1} + C'(3) \cdot \omega_0^{-1})$$

Since  $\omega_0^{-1} = 1$ , this becomes:

$$C(0) = \frac{1}{4} (6 + (1 + 3i) + 0 + (1 - 3i)) = \frac{1}{4} (6 + 1 + 3i + 1 - 3i)$$

$$C(0) = \frac{1}{4} \cdot 8 = 2$$

#### Step 3: Calculate $C(2)$

$$C(2) = \frac{1}{4} (C'(0) \cdot \omega_2^{-2} + C'(1) \cdot \omega_2^{-2} + C'(2) \cdot \omega_2^{-2} + C'(3) \cdot \omega_2^{-2})$$

Since  $\omega_2^{-2} = -1$ , this simplifies to:

$$C(2) = \frac{1}{4} (6 - (1 + 3i) + 0 - (1 - 3i))$$

Simplify further:

1.  $-(1 + 3i) = -1 - 3i$
2.  $-(1 - 3i) = -1 + 3i$

So:

$$C(2) = \frac{1}{4} (6 - 1 - 3i - 1 + 3i) = \frac{1}{4} \cdot 4 = 1$$

## Final Result

Thus, the coefficients of the polynomial  $C(x) = 2 + 3x + x^2$  are:

$$[2, 3, 1]$$

where:

- $C(0) = 2$  (constant term),
- $C(1) = 3$  (coefficient of  $x$ ),
- $C(2) = 1$  (coefficient of  $x^2$ ).

The Inverse FFT has given us the coefficients in the time domain, confirming that:

$$C(x) = 2 + 3x + x^2$$