CSC263H
Data Structures and Analysis

Prof. Bahar Aameri & Prof. Marsha Chechik

Winter 2024, Week 11

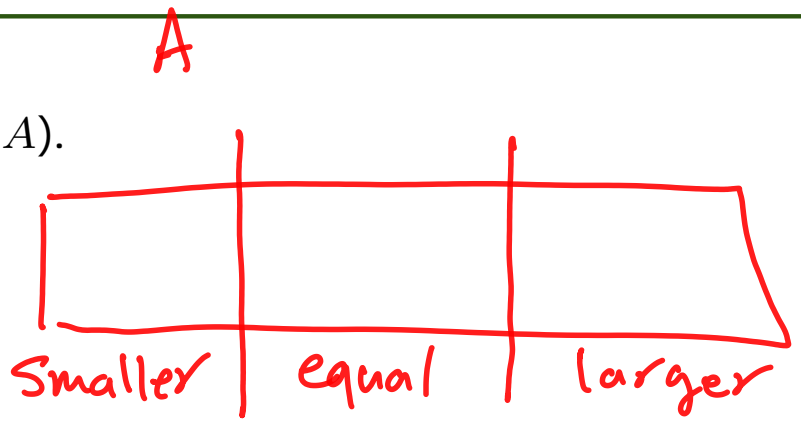
Today's Lecture

- Review of QuickSort
- QuickSort Run Time Analysis: Worst Case
- QuickSort Run Time Analysis: Average Case
- Randomized QuickSort
- Randomized Algorithms

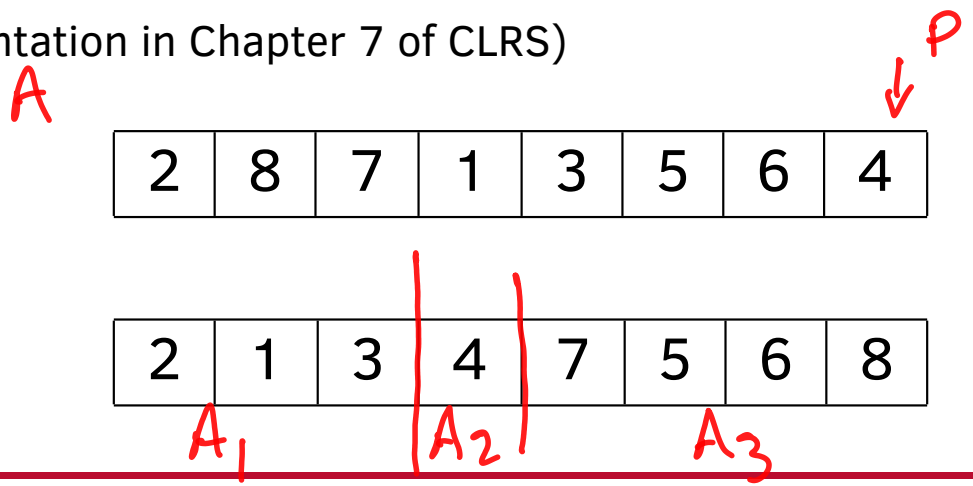
QuickSort: The Idea

Input: An array A of numbers with size n.
output: Return A s.t elements in A are sorted in non-decreasing order.

1. If $len(A)$ is 1, return.
2. Pick a pivot p (usually the last element of A).
3. Partition the array into:
   - A_1: elements of A less than p.
   - A_2: elements of A equal to p.
   - A_3: elements of A greater than p.
4. Recursively repeat this procedure for A_1 and A_3.

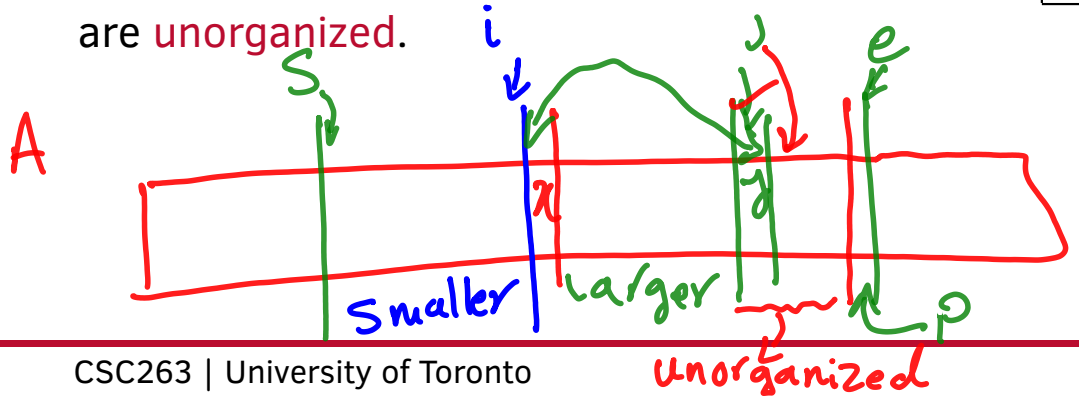


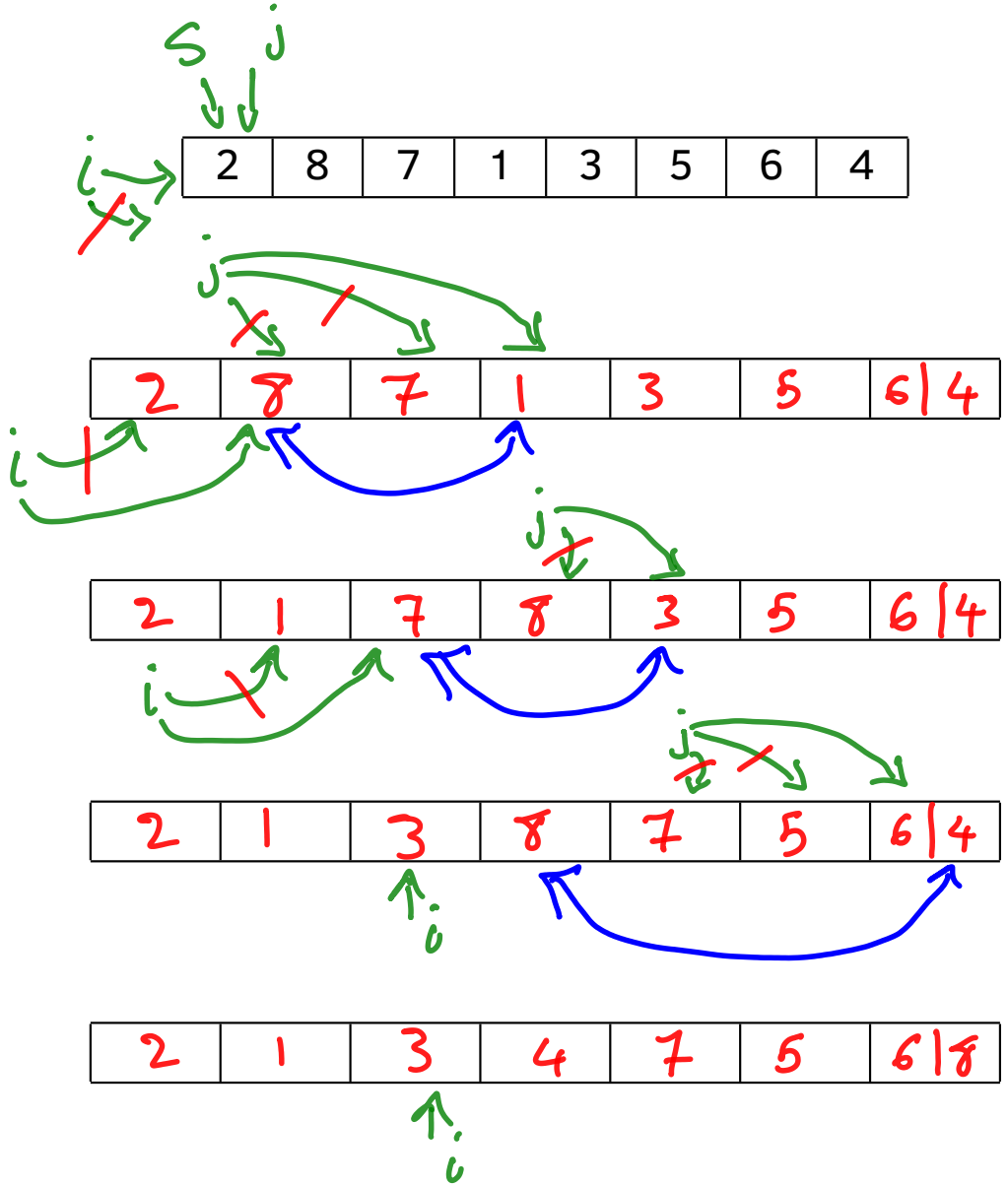
(Complete implementation in Chapter 7 of CLRS)



Partition(A, s, e):
1. $p = A[e]$
2. $i = s - 1, j = s$
3. **while** $j \leq e - 1$:
4. **if** $A[j] \leq p$
5. $i = i + 1$
6. exchange $A[i]$ with $A[j]$
7. $j = j + 1$
8. exchange $A[i + 1]$ with $A[e]$
9. **return** $i + 1$

After every iteration, items stored:
- from $A[s]$ to $A[i]$ (inclusive) are less than or equal to p.
- from $A[i + 1]$ to $A[j - 1]$ (inclusive) are larger than p.
- from $A[j]$ to $A[e - 1]$ (inclusive) are unorganized.





x>p
y>p

QuickSort: Worst-case Running Time

Let $T(n)$ denotes the **worst-case** run time of QuickSort for an input A of size n.

- We choose **the number of comparisons** (Line #4) performed during QuickSort as the **representative operation**.

QuickSort: Worst-case Running Time

Observations:

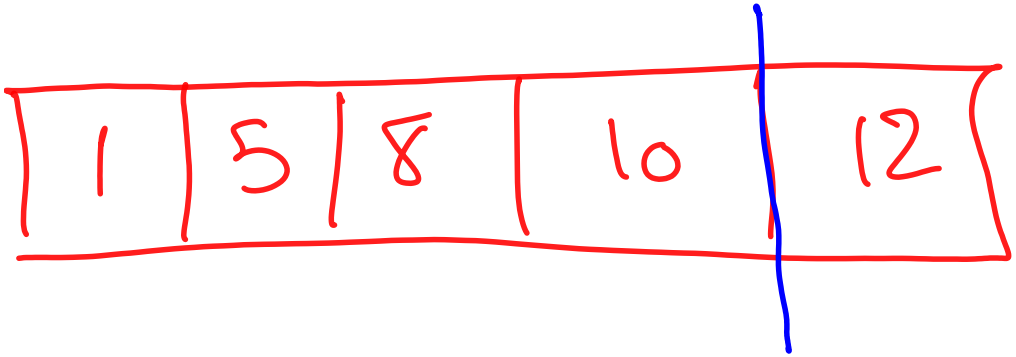
1. A pivot never goes into a sub-array on which a recursive call is made. That is, **each element** in A can be chosen as **pivot** at **most once**.
2. In **each recursive call,** elements in the sub-array are **only compared to pivots**.
3. After being a pivot, that **pivot** will **never be compared** with anyone anymore.

Implication: **Every pair** (a, b) in A, are compared with each other **at most once**.

Conclusion: The total number of comparisons is less than or equal to the **total number of pairs**.

Number of pairs in an array of size n: $\binom{n}{2} = \frac{n(n-1)}{2} \Rightarrow T(n) \in O(n^2)$ (1)

QuickSort: Running Time Analysis



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

$\Rightarrow T(n) \in \Omega(n^2)$ ②

①, ② $\Rightarrow T(n) \in \Theta(n^2)$

How is QuickSort "quick" then??

It is quick in the **average case**.

Computing **Average-case** Run Time for A – Review

1. Define S_n: space of **all inputs** of size n.
2. Assume a **probability distribution** over S_n: specifying likelihood of each input.
3. Define the **random variable** t_n over S_n:
   $t_n(x)$: number of steps executed by the algorithm on an input x in S_n.
4. **Compute the expected value** $\mathbb{E}[t_n]$ of $t_n(x)$.

Indicator Random Variables – Review

If $X = X_1 + X_2 + ... + X_m$, then

$$\begin{aligned}\mathbb{E}[X] &= \mathbb{E}[X_1] + ... + \mathbb{E}[X_m] \\ &= Pr[X_1 = 1] + ... + Pr[X_m = 1]\end{aligned}$$

1. S_n: All permutations of n numbers.
   (Assumption: all elements are distinct)

2. **Probability Assumption:** each permutation appears equally likely.

3. $t_n(A)$: number of comparisons performed on an array A of size n.

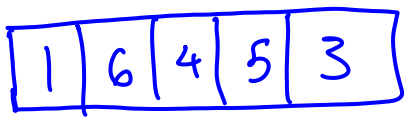
4. Compute $\mathbb{E}[t_n]$.

Let $z_1, z_2, z_3, \ldots, z_n$ be the sequence of elements in A in the **sorted order**.
We define the following indicator random variable:

$X_{i,j} = X_{j,i}$

$$X_{i,j} = \begin{cases} 1 & \text{if the values } z_i \text{ and } z_j \text{ are compared} \\ 0 & \text{otherwise.} \end{cases}$$

A



1, 3, 4, 5, 6

z_1 z_2 z_3 z_4 z_5

$$t_n = \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}$$

$$\mathbb{E}[t_n] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E}[X_{i,j}] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[z_i \text{ is compared to } z_j]$$

Computing $Pr(z_i$ is compared with $z_j)$:

Observations:

1. z_i and z_j are compared when
   (1) one of them is chosen as the pivot
   (2) the other is in the same partition

2. z_i and z_j are in different partitions if a number z_k, ($z_i < z_k < z_j$), is chosen as a pivot before z_i and z_j is chosen.

3. So z_i and z_j are compared by QuickSort iff one of z_i or z_j is selected as pivot before $z_{i+1}, z_{i+2}, \ldots, z_{j-1}$.

Conclusion: z_i and z_j are compared by QuickSort iff out of the numbers $\{z_i, z_{i+1}, z_{i+2}, \ldots, z_j\}$, one of them is selected to be pivot first.

$Pr(z_i$ is compared with $z_j)$
= $Pr(z_i$ or z_j is the first element among $\{z_i, z_{i+1}, z_{i+2}, \ldots, z_j\}$ chosen as pivot)

$= \frac{2}{j-i+1}$

$$\mathbb{E}[t_n] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} Pr(z_i \text{ is compared with } z_j)$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= 2(n+1) \underbrace{\sum_{k=1}^{n-1} \frac{1}{k+1}}_{\in \Theta(\lg n)} - 2(n-1)$$

$$\Rightarrow E[t_n] \in \Theta(n \lg n)$$

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = 2 \sum_{i=1}^{n} \sum_{K=1}^{n-i} \frac{1}{K+1}$$

(change of index: $K = j - i$)

$$= \sum_{K=1}^{n-1} \frac{1}{K+1} + \sum_{K=1}^{n-2} \frac{1}{K+1} + \cdots + \sum_{K=1}^{n-(n-1)} \frac{1}{K+1}$$

$K=1 \rightarrow (n-1) \times 1$

$K=2 \rightarrow (n-2) \times 1$

$K=3 \rightarrow (n-3) \times 1$

$\vdots$

$K=n-1 \rightarrow 1 \times 1$

$$= 2 \sum_{K=1}^{n-1} \frac{n-K}{K+1}$$

$$= 2 \sum_{K=1}^{n-1} \left(\frac{n+1}{K+1} - 1 \right)$$

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

Randomized QuickSort

Problem:

- Average-case analysis works only if **each permutation is equally likely**. In practice, this may not be true:
  - It is often impossible to know what the input distribution really is.
  - If the person who provides the inputs is malicious, they can provide worst-inputs and cause worst-case runtime.

Solution:

- Average-case analysis works only if **each permutation is equally likely**. In practice, this may not be true:
- Shuffle the input array **uniformly randomly**.
  Implication: After shuffling, the arrays look like drawn from a uniform distribution.

Hence: The assumption in the average-case analysis true.
Average-case $\Theta(n \log n)$ is guaranteed.

Randomized Algorithms

- **Las Vegas Algorithm:** The solutions generated by the algorithm is guaranteed to be correct, but runtime depends on random choices.
  Example:
  - Randomized QuickSort.
  - Another way of **randomizing QuickSort**: Choose **pivot** randomly (read Sections 7.3 and 7.4 of David Liu's notes for details).

- **Monte Carlo Algorithm:** Runtime of the algorithm is deterministic, but the output is based on random choices.

After Lecture

- After-lecture Readings: Chapter 5 of the course notes, Chapter 7 of CLRS.
- Problems at the end of Chapter 5 of the course notes.
- Problems 7.1-1, 7.1-4, 7.2-2, 7.2-3 in CLRS.