CS 457, Fall 2016

Drexel University, Department of Computer Science

Lecture 7

Quicksort (Running Time)

QUICKSORT (*A*, *p*, *r*)

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

PARTITION (*A*, *p*, *r*)

```
1.  x = A[r]
2.  i = p – 1
3.  for j = p to r – 1
4.      if A[ j] <= x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.  exchange A[i+1] with A[r]
8.  return i+1
```

<u>Lemma</u>:
Let $\boldsymbol{X}$ be the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT on an n-element array.
Then the running time of QUICKSORT is $\boldsymbol{O(n+X)}$.

Quicksort (Running Time)

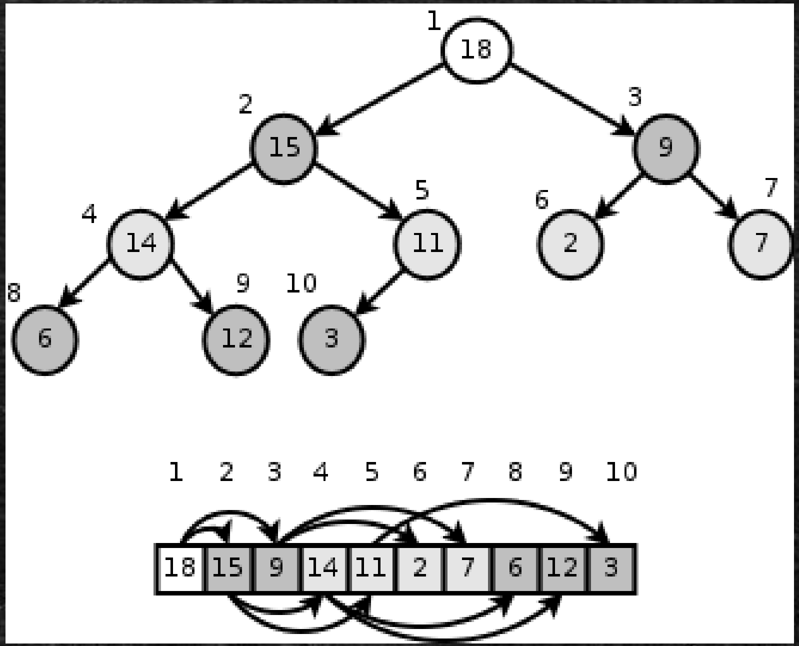
- Denote the sorted elements of the array by $z_1, z_2, \ldots, z_n$
- Let $X_{ij} = \mathbb{I}\{z_i \text{ is compared to } z_j\}$
- Then, $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$ and $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr\{z_i \text{ is compared to } z_j\}$

- Let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$
- $Pr\{z_i \text{ is compared to } z_j\}$ = $Pr\{z_i \text{ or } z_j \text{ is first pivot chosen from } Z_{ij}\}$ = $\frac{2}{j-i+1}$
- So, $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$ = $\sum_{i=1}^{n-1} \sum_{k=1}^{n-1} \frac{2}{k}$ = $\sum_{i=1}^{n-1} O(\log n)$ = $O(n \log n)$

Heapsort

Heap data structure:

PARENT (i)
    return $\lfloor i/2 \rfloor$
LEFT (i)
    return $2i$
RIGHT (i)
    return $2i + 1$



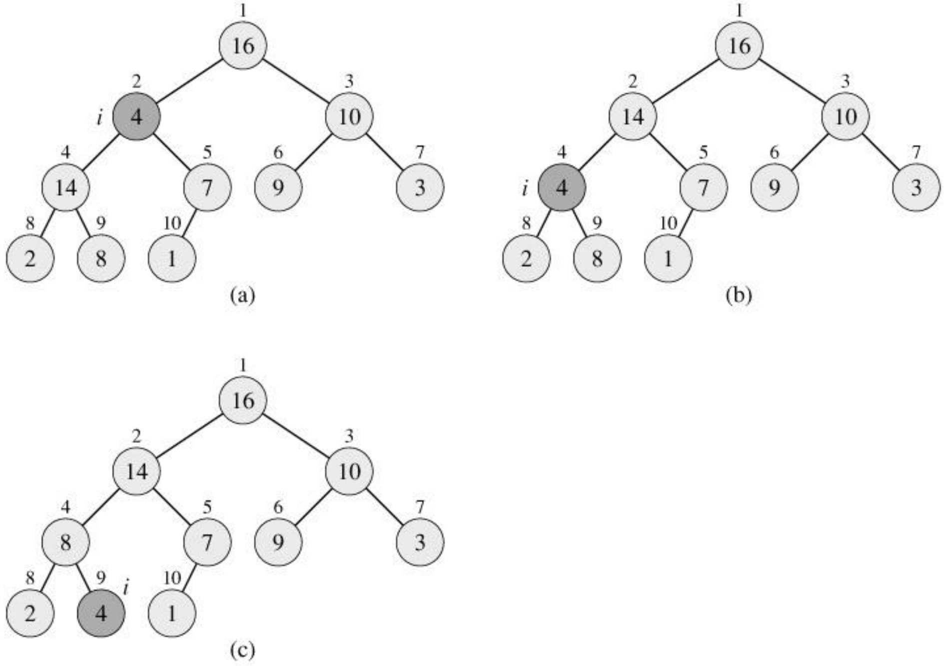
Max-heap property: $A[\text{PARENT}(i)] \geq A[i]$

Heapsort

Max-Heapify (A, i)

```
l = left [i]
r = right [i]
if  l ≤ A.heap-size and A[l] > A[i]
        largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
        largest = r
if largest  ≠ i
        exchange A[i] with A[largest]
        Max-Heapify (A, largest)
```

Call to **Max-Heapify (A,2)**



Today's Lecture

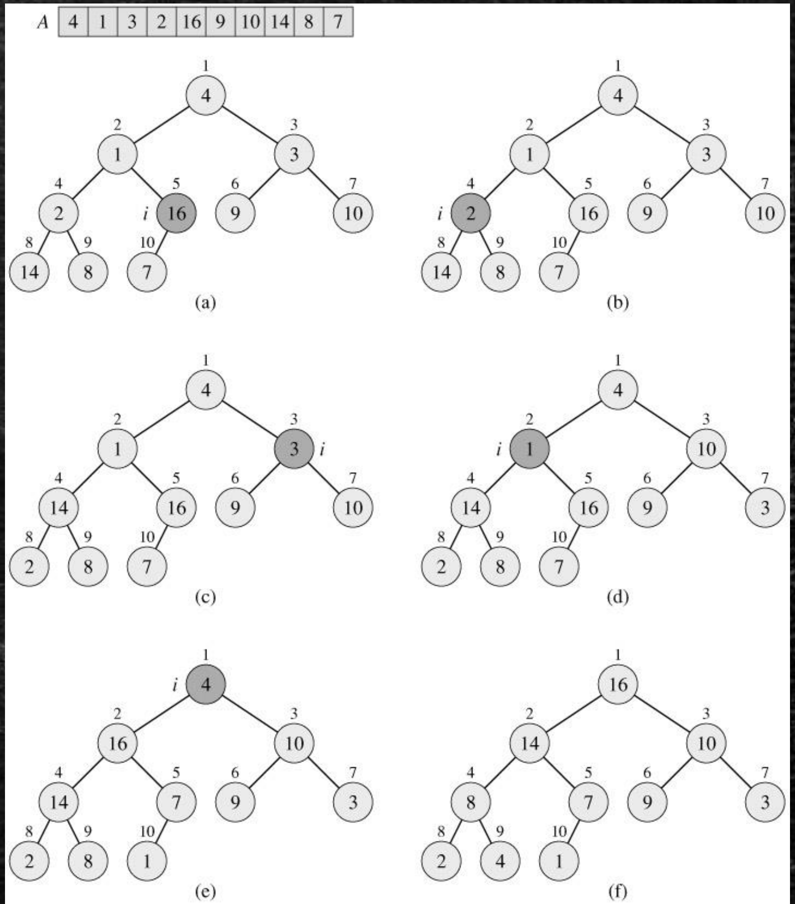
- Heapsort
- Priority Queues (Read from textbook)
- Selection Problem

Heapsort

Build-Max-Heap (A)

1. A.heap-size = A.length
2. **for** $i = \lfloor \text{A.length}/2 \rfloor$ **down to** 1
3. Max-Heapify(A, i)

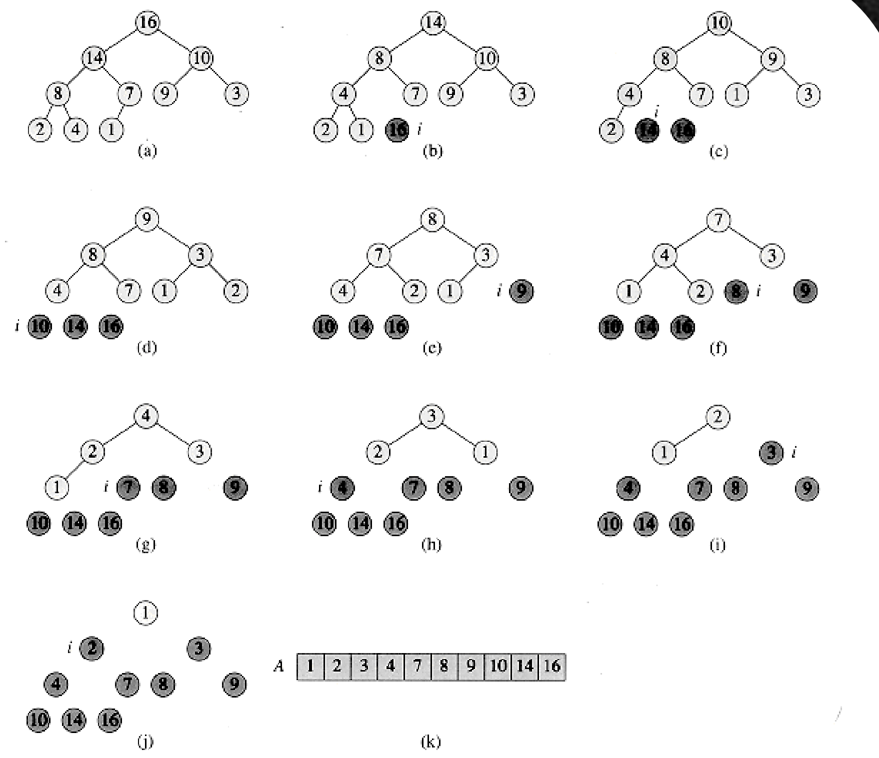
$$\sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \right) = O(n)$$



Heapsort

Heapsort(A)

1. Build-Max-Heap(A)
2. **for** i =A.length **down to** 2
3. exchange A[1] with A[i]
4. A.heap-size = A.heap-size -1
5. Max-Heapify(A,1)



Order Statistics

The i^{th} **order statistic** of a set of n numbers: the i^{th} smallest number in sorted sequence:

A	4	1	3	2	16	9	10	14	8	7

- **Minimum** or **first order statistic**: 1
- **Maximum** or n^{th} **order statistic**: 16
- **Median** or $(n/2)^{\text{th}}$ **order statistic**: 7 or 8
  (both are medians, happens when n is even!)

The Selection Problem

- **Input:** An array $\mathbf{A}$ of distinct numbers of size n, and a number $\boldsymbol{i}$.
- **Output:** The element $\boldsymbol{x}$ in $\mathbf{A}$ that is larger than exactly $\boldsymbol{i}-\mathbf{1}$ other elements in $\mathbf{A}$.

- Finding ***maximum*** and ***minimum*** can be easily solved in linear time (i.e. $O(n)$). (it' s actually $\Theta(n)$).

Finding minimum

Minimum(A)

```
min = A[1]
for i = 2 to A.length
    if min > A[i]
        min = A[i]
return min
```

- Makes n-1 comparisons
- Can we do any better?

Finding min and max simultaneously

- How fast can you find both?

Finding min and max simultaneously

- By making 2n-2 comparisons
  - Find min making n-1 comparisons
  - Find max making n-1 comparisons

- Compare each pair of elements
  - 1 comparison per pair to determine which one is small/big
- Compare the smaller of each pair with the global min
- Compare the larger of each pair with the global max
  - 2 comparisons per pair
- 3n/2 comparisons in total!

Finding i^{th} element

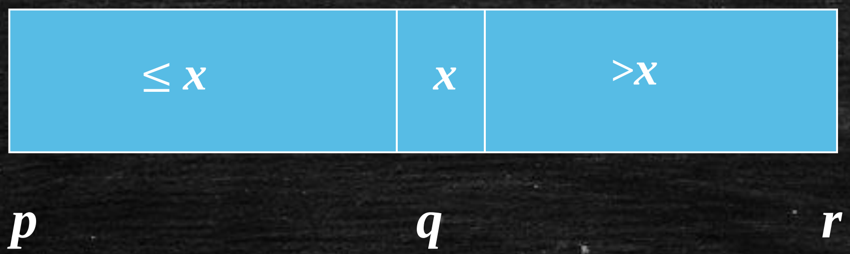
- Propose an algorithm that computes the i^{th} element

Finding i^{th} element

- Sort the array ***A***, and return the entry in $\boldsymbol{i}^{th}$ position:
  - Sorting ***A*** takes $O(n \log n)$.
  - The $\boldsymbol{i}^{th}$ entry can then be returned in constant time.
- Worst case running time
  - $O(n \log n)$
- Can we do better?

A Randomized Selection Algorithm

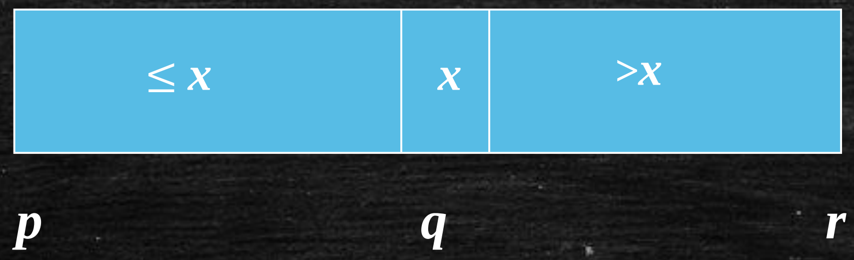
- Think about the properties of **Partition( )** algorithm:



- If $i == q$, then we have x as the i^{th} order statistic.
- What if this not the case?

A Randomized Selection Algorithm

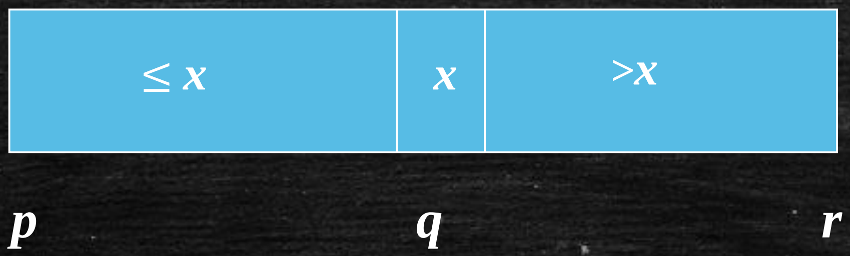
- If $i < q$, we look for the i^{th} order statistic among first ***q-1*** elements:



- We can call **Partition( )**, with parameters ***(A,p,q-1)***

A Randomized Selection Algorithm

- If $i > q$, we look for the i^{th} order statistic among the elements in $[q + 1, r]$



- We can call **Partition( )**, with parameters ***(A,q+1,r)***

Randomized Selection Algorithm

Randomized-Select(A, p, r, i)

```
if p == r
    return A[p]
q = Randomized-Partition(A, p, r)
k = q - p + 1
if i == k
    return A[q]
else if i ≤ k
    Randomized-Select(A, p, q - 1, i)
else
    Randomized-Select(A, q + 1, r, i - k)
```

Running Time

- What is the worst-case running time of the algorithm?
  - a) $O(n^2)$
  - b) $O(n \log n)$
  - c) $O(n)$

- What about the average-case?
  - a) $O(n^2)$
  - b) $O(n \log n)$
  - c) $O(n)$

Running Time

- $X_k = \mathbb{I}\{\text{the subarray } A[p, \ldots, q] \text{ has exactly } k \text{ elements}\}$
- $\mathbb{E}[X_k] = \frac{1}{n}$

$$
\begin{aligned}
T(n) \le & \sum_{k=1}^{n} X_k \left(T(\max\{k-1, n-k\}) + O(n)\right) \\
= & \sum_{k=1}^{n} X_k \left(T(\max\{k-1, n-k\})\right) + \sum_{k=1}^{n} X_k\, O(n) \\
= & \sum_{k=1}^{n} X_k \left(T(\max\{k-1, n-k\})\right) + O(n)
\end{aligned}
$$

Running Time

$$
\begin{aligned}
\mathbb{E}[T(n)] \le\ & \mathbb{E}\left[\sum_{k=1}^{n} X_k \left(T(\max\{k-1, n-k\})\right) + O(n)\ \right] \\
&= \sum_{k=1}^{n} \mathbb{E}[X_k(T(\max\{k-1, n-k\}))]\ + O(n) \\
&= \sum_{k=1}^{n} \mathbb{E}[X_k]\ \mathbb{E}[T(\max\{k-1, n-k\})]\ + O(n) \\
&= \sum_{k=1}^{n} \frac{1}{n}\ \mathbb{E}[T(\max\{k-1, n-k\})]\ + O(n)
\end{aligned}
$$

Running Time

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

and $\max\{k-1, n-k\}) = \begin{cases} k-1 \text{ if } k > \lceil n/2 \rceil \\ n-k \text{ if } k \le \lceil n/2 \rceil \end{cases}$

So, $\mathbb{E}[T(n)] \le \frac{2}{n} \sum_{k=\left\lfloor \frac{n}{2} \right\rfloor}^{n-1} \mathbb{E}[T(k)] + O(n)$

Running Time

Using substitution, we solve: $\mathbb{E}[T(n)] \leq \frac{2}{n}\sum_{k=\left\lfloor\frac{n}{2}\right\rfloor}^{n-1} \mathbb{E}[T(k)] + O(n)$

Assume: $\mathbb{E}[T(n')] \leq cn'$ for some constant c and every $n' < n$.

Then, $\mathbb{E}[T(n)] \leq \frac{2}{n}\sum_{k=\left\lfloor\frac{n}{2}\right\rfloor}^{n-1} ck + an \leq \frac{2c}{n}\left(\frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2}\right) + an$

Which leads to $\mathbb{E}[T(n)] \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$

Running Time

Using substitution, we solve: $\mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} \mathbb{E}[T(k)] + O(n)$

Assume: $\mathbb{E}[T(n')] \leq cn'$ for some constant c and

Then, $\mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} ck + an \leq \frac{2c}{n}\left(\frac{n^2 - n}{2} - \right.$ … an

for $\boldsymbol{c > 4a}$,

and $\boldsymbol{n \geq \frac{2c}{c-4a}}$

Which leads to $\mathbb{E}[T(n)] \leq cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right) \leq cn$

Selection Problem

- Can we design algorithms with better **worst case** guarantees?

Worst case linear time selection

Select(A,p,r,i)

1. Divide **A** into $\boldsymbol{n/5}$ groups of size $\mathbf{5}$.
2. Find the median of each group of $\mathbf{5}$ by brute force, and store them in a set $\mathbf{A'}$ of size $\boldsymbol{n/5}$.
3. Recursively use **Select**$(\mathbf{A'},\ \mathbf{1},\ \boldsymbol{n/5}, \boldsymbol{n/10})$ to find the median $\mathbf{x}$ of $\boldsymbol{n/5}$ medians.
4. Partition elements of **A** around $\mathbf{x}$. Let $\boldsymbol{k}$ be the order of $\mathbf{x}$ found in the partitioning.
5. if $\boldsymbol{i = k}$
6. return $\mathbf{x}$
7. else if $\boldsymbol{i < k}$
8. **Select**$(\mathbf{A},\ \mathbf{p},\ \mathbf{q-1},\ \boldsymbol{i})$
9. else
10. **Select**$(\mathbf{A}, \mathbf{q+1}, \mathbf{r}, \boldsymbol{i-k})$