Chapter 9
Medians and Order Statistics

Sultan ALPAR
associate professor, IITU
s.alpar@iitu.edu.kz

Outline

- **Minimum and maximum**
- Selection in expected linear time
- Selection in worst-case linear time

Order statistics

- The ith **order statistic** of a set of n elements is the ith smallest element.
- The **minimum** is the first order statistic ($i = 1$).
- The **maximum** is the nth order statistic ($i = n$).
- A **median** is the "halfway point" of the set.
- When n is odd, the median is unique, at $i = (n + 1)/2$.
- When n is even, there are two medians:
  - The **lower median**: $i = \lfloor (n+1)/2 \rfloor$
  - The **upper median**: $i = \lceil (n+1)/2 \rceil$
  - We mean lower median when we use the phrase "the median".

The selection problem

- How can we find the ith order statistic of a set and what is the running time?
- **Input:** A set A of n (distinct) number and a number i, with $1 \le i \le n$.
- **Output:** The element $x \in A$ that is larger than exactly $i - 1$ other elements of A.
- The **selection problem** can be solved in $\boldsymbol{O(n\lg n)}$ time.
  - Sort the numbers using an $O(n\lg n)$-time algorithm, such as heapsort or merge sort.
  - Then return the ith element in the sorted array.
- Are there faster algorithms?
  - An $O(n)$-time algorithm would be presented in this chapter.

Finding minimum

- We can easily obtain an upper bound of $n-1$ comparisons for finding the minimum of a set of n elements.
  - Examine each element in turn and keep track of the smallest one.
  - The algorithm is optimal, because each element, except the minimum, must be compared to a smaller element at least once.

```
MINIMUM(A)
    min ← A[1]
    for i ← 2 to length[A]
        do if min > A[i]
            then min ← A[i]
    return min
```

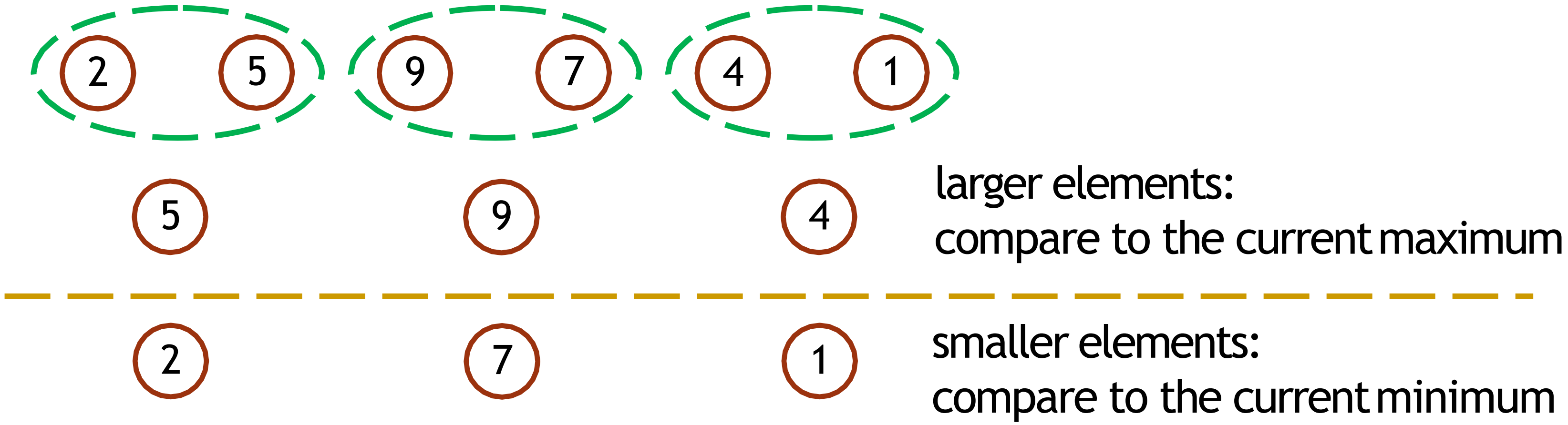
- The maximum can be found in exactly the same way by replacing the > with < in the above algorithm.

Simultaneous minimum and maximum

- Some applications need both the minimum and maximum.
  - Find the minimum and maximum independently, using $n-1$ comparisons for each, for a total of $2n-2$ comparisons.
- In fact, at most $3n/2$ comparisons are needed:
  - Maintain the minimum and maximum of elements seen so far.
  - Process elements in pairs.
  - Compare the elements of a pair to each other.
  - Then compare the larger element to the maximum so far, and compare the smaller element to the minimum so far.
- This leads to only 3 comparisons for every 2 elements.

Simultaneous minimum and maximum

- An observation
  - If we compare the elements of a pair to each other, the larger can't be the minimum and the smaller can't be the maximum.
  - So we just need to compare the larger to the current maximum and the smaller to the current minimum.
  - It costs 3 comparisons for every 2 elements.
    - The previous method costs 2 comparisons for each element.



Simultaneous minimum and maximum

- Setting up the initial values for the min and max depends on whether *n* is odd or even.
  - If *n* is even, compare the first two elements and assign the larger to max and the smaller to min.
  - If *n* is odd, set both min and max to the first element.
- If *n* is even, # of comparisons $= \frac{3(n-2)}{2} + 1 = \frac{3n}{2} - 2.$
- If *n* is odd, # of comparisons $= \frac{3(n-1)}{2} = 3\lfloor n/2 \rfloor$
- In either case, the # of comparisons is $\le 3\lfloor n/2 \rfloor$

Outline

- Minimum and maximum
- **Selection in expected linear time**
- Selection in worst-case linear time

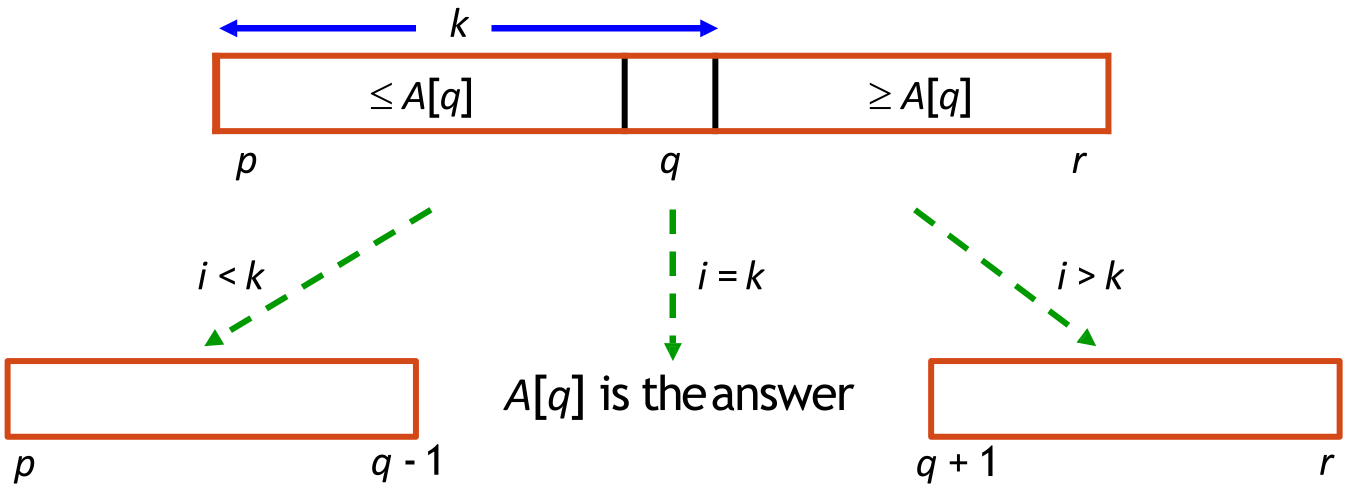
Selection in expected linear time

- In fact, selection of the *i*th smallest element of the array *A* can be done in $\Theta(n)$ time.
- We first present a randomized version in this section and then present a deterministic version in the next section.
- The function RANDOMIZED-SELECT:
  - is a divide-and-conquer algorithm,
  - uses RANDOMIZED-PARTITION from the quicksort algorithm in Chapter 7, and
  - recurses on one side of the partition only.

RANDOMIZED-SELECT procedure

```
RANDOMIZED-SELECT(A, p, r, i)
   if p = r
      then return A[p]
   q ← RANDOMIZED-PARTITION(A, p, r)
   k ← q − p + 1
   if i = k          /* the pivot value is the answer */
      then return A[q]
   elseif i < k
      then return RANDOMIZED-SELECT(A, p, q − 1, i)
   else return RANDOMIZED-SELECT(A, q, r, i − k)
```

RANDOMIZED-SELECT procedure



To find the ***i*th** order statistic in $A[p \ldots q-1]$

To find the **(*i*–*k*)th** order statistic in $A[q+1 \ldots r]$

Algorithm analysis

- The **worst case:** always recurse on a subarray that is only 1 element smaller than the previous subarray.
  - $T(n) = T(n-1) + \Theta(n)$
    $= \Theta(n^2)$
- The **best case:** always recurse on a subarray that has half of the elements smaller than the previous subarray.
  - $T(n) = T(n/2) + \Theta(n)$
    $= \Theta(n)$ (Master Theorem, case3)

Algorithm analysis

- The **average case:**
  - We will show that $T(n) = \Theta(n)$.
  - For $1 \leq k \leq n$, the probability that the subarray $A[p .. q]$ has k elements is $1/n$.
  - To obtain an upper bound, we assume that $T(n)$ is monotonically increasing and that the ith smallest element is always in the larger subarray.
  - So, we have

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

$$T(n) \leq - \sum_{k=1}^{n} (T(\max(k-1, n-k) + O(n)) = - \sum_{k=1}^{n} (T(\max(k-1, n-k))) + O(n)$$

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

Algorithm analysis

- Solve this recurrence by substitution:
  - Assume $T(n) \le cn$ for sufficiently large c.
  - The function described by the $O(n)$ term is bounded by an for all $n > 0$.
  - Thus, if we assume that
  - $T(n) = O(1)$ for
  - $n < 2c/(c - 4a)$,
  - we have $T(n) = O(n)$.

$$
\begin{aligned}
\mathrm{E}[T(n)] &\le \frac{2}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} ck + an \\
&= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2\rfloor-1} k\right) + an \\
&= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2\rfloor - 1)\lfloor n/2\rfloor}{2}\right) + an \\
&\le \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) + an \\
&= \frac{2c}{n}\left(\frac{n^2-n}{2} - \frac{n^2/4 - 3n/2 + 2}{2}\right) + an \\
&= \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + an \\
&= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an \\
&\le \frac{3cn}{4} + \frac{c}{2} + an \\
&= cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right) .
\end{aligned}
$$

Outline

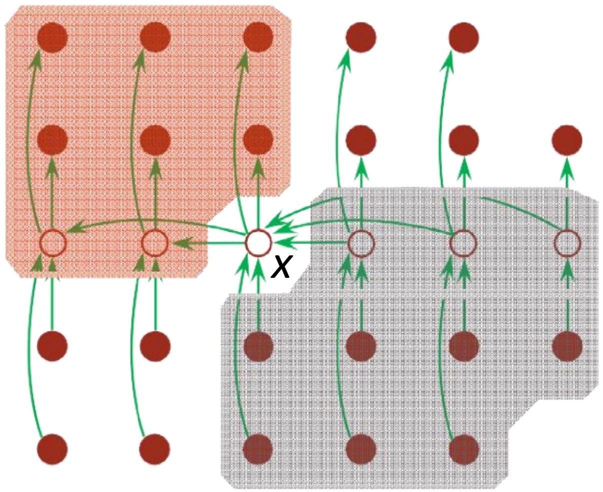
- Minimum and maximum
- Selection in expected linear time
- **Selection in worst-case linear time**

SELECT algorithm

- **Idea:** Guarantee a good split when the array is partitioned.
- The SELECT algorithm:
  1. Divide n elements into groups of 5 elements.
  2. Find median of each of the $n/5$ groups.
     - Run insertion sort on each group.
     - Then just pick the median from each group.
  3. Use SELECT recursively to find median x of the $n/5$ medians.
  4. Partition the n elements around x.
     - Let x be the kth element of the array after partitioning.
     - There are $k-1$ elements on the low side of the partition and $n-k$ elements on the high side.

SELECT algorithm

5. Now there are three possibilities:
   - If $i = k$, then return x.
   - If $i < k$, then use SELECT recursively to find ith smallest element on the low side.
   - If $i > k$, then use SELECT recursively to find $(i-k)$th smallest element on the high side.



○ : The median of a group.

→ : From larger to smaller.

: Elements in this region are greater than x.

: Elements in this region are samller than x.

Time complexity

- At least half of the medians are $\geq x$.
  - Precisely, at least $\lceil n/5 \rceil / 2$ medians $\geq x$.
- These group contributes 3 elements that are > x, except for 2 of the groups:
  - the group containing x, and
  - the group with < 5 elements.
- The number of elements greater than x is at least:
  - $3\ (\lceil n/5 \rceil / 2 - 2) \geq \mathbf{3n/10 - 6}$.
- Similarly, at least $3n/10 - 6$ elements are less than x.
- Thus, SELECT is called recursively on $\leq \mathbf{7n/10 + 6}$ elements in step 5.

Time complexity

- The SELECT algorithm:
  1. Divide n elements into groups of 5 elements. $O(n)$
  2. Find median of each of the $n/5$ groups. $O(n)$
     - Run insertion sort on each group.
     - Then just pick the median from each group.
  3. Use SELECT recursively to find median x of the $n/5$ medians. $T(n/5)$
  4. Partition the n elements around x. $O(n)$
     - Let x be the kth element of the array after partitioning.
     - There are $k-1$ elements on the low side of the partition and $n-k$ elements on the high side.
  5. Now there are three possibilities: $T(7n/10+6)$
     - If $i = k$, then return x.
     - If $i < k$, then use SELECT recursively to find i th smallest element on the low side.
     - If $i > k$, then use SELECT recursively to find $(i-k)$th smallest element on the high side.
- Time complexity: $T(n) \le T(n/5) + T(7n/10+6) + O(n)$.

Time complexity

- Solve this recurrence by substitution:
  - Assume $T(n) \le cn$ for sufficiently large c.
  - The function described by the $O(n)$ term is bounded by an for all $n > 0$.
  - Then, we have

$$
\begin{aligned}
T(n) &\le c\,n/5 + c(7n/10+6) + an \\
&\le cn/5 + c + 7cn/10 + 6c + an \\
&= 9cn/10 + 7c + an \\
&= cn + (-cn/10 + 7c + an)
\end{aligned}
$$

  - This last quantity is $\le cn$ if we choose $c \ge 20a$.

$$
\begin{aligned}
-cn/10 + 7c + an &\le 0 \\
cn/10 - 7c &\ge an \\
c(n-70) &\ge 10an \\
c &\ge 10a(n/(n-70))
\end{aligned}
$$

Notice: $(n/n\text{-}70) \le 2$ for $n \ge 140$.

Conclusion

- Thus, the running time is linear because these algorithms do not sort;
- The linear-time behavior is not a result of assumptions about the input, as was the case for the sorting algorithms in Chapter 8.
- Sorting requires $\Omega(n \lg n)$ time in the comparison model, even on average, and thus the method of sorting and indexing presented in the introduction to this chapter is asymptotically inefficient.