Order Statistics

- The \( i^{th} \) smallest of \( n \) elements is the \( i^{th} \) order statistic:
  - Minimum: \( i = 1 \)
  - Maximum: \( i = n \)
  - Median: \( i = \frac{n}{2} \) (if \( n \) is odd)
  - Median: \( i = \frac{n+1}{2} \) (if \( n \) is even)

- What is a naive algorithm for finding the \( i^{th} \) order statistic of \( n \) (distinct) numbers with \( 1 \leq i \leq n \)?
  - Heapsort or Mergesort
  - Return the \( i^{th} \) element

- What is its worst-case running time?
  \[ O(n \log (n)) + O(1) = O(n \log n) \]
Minimum (Maximum similar)

- MINIMUM(A,n) // or MINIMUM(A) then use A.length

```
1 min = A[1]
2 for i = 2 to n
3   if min > A[i]
4     min = A[i]
5 return min
```

- How many comparisons are needed?

- O? Ω? Θ?

Max and Min

- How many comparisons are needed to find Max and Min independently?

- Can we do better?
Example: Simultaneous Max and Min

- n = 6 (even), A = <2, 4, 8, 5, 3, 1>
- Consider first pair, set largest to 4, smallest to 2

- For each pair of elements, how many comparisons?
- How does this compare to previous method?

Example: Simultaneous Max and Min

- n = 5 (odd), A = <2, 7, 1, 3, 4>
Analysis: Simultaneous Max and Min

• Total number of comparisons when:
  • n is even:
    - Moral: at most $\frac{3n}{2}$ comparisons are needed

• n is odd:

Selection in linear time

• From previous argument, can determine min ($i = 1$) and max ($i = n$) either separately or simultaneously in linear time $\Theta(n)$

• More generally, we argue we can find the $i^{th}$ order statistic of $A(p,r)$ in linear time using a randomized algorithm.

• Idea of $\text{RANDOMIZED-SELECT}(A, p, r, i)$:
  • Use $\text{RANDOMIZED-PARTITION}$ from quicksort to partition $A$ so that $A[q]$ is in the correct position of the array. If $q = i$, done, otherwise recurse on only ONE side of the partition
### RANDOMIZED-SELECT(A, p, r, i)

\[ q = \text{RANDOMIZED-PARTITION}(A, p, r) \]

Note that the index \( q \) corresponds to the \( k \)th order statistic of this array where \( k = q - p + 1 \)

```
if i < k
    follow this branch to find \( i \)th order statistic in \( A[p, q-1] \)
else
    follow this branch to find the \((i-k)\)th order statistic in \( A[q+1, r] \)
```

### Randomized-Partition

\[ \text{RANDOMIZED-PARTITION}(A, p, r) \text{ from QuickSort (p. 179)} \]

---

### Order Statistics: Input \( A[p...r] \), find \( i \)th

- \( \text{RANDOMIZED-SELECT}(A, p, r, i) \)

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if ( p = r )</td>
</tr>
<tr>
<td>2</td>
<td>return ( A[p] )</td>
</tr>
<tr>
<td>3</td>
<td>( q = \text{RANDOMIZED-PARTITION}(A, p, r) )</td>
</tr>
<tr>
<td>4</td>
<td>( k = q - p + 1 )</td>
</tr>
<tr>
<td>5</td>
<td>if ( i = k ) // So pivot value is it!</td>
</tr>
<tr>
<td>6</td>
<td>return ( A[q] )</td>
</tr>
<tr>
<td>7</td>
<td>else if ( i &lt; k )</td>
</tr>
<tr>
<td>8</td>
<td>return ( \text{RANDOMIZED-SELECT}(A, p, q-1, i) )</td>
</tr>
<tr>
<td>9</td>
<td>else return ( \text{RANDOMIZED-SELECT}(A, q+1, r, i-k) )</td>
</tr>
</tbody>
</table>
Example

- A: <6, 10, 13, 5, 8, 3, 2, 11>

Selection in worst-case Linear Time

- The worst-case for RANDOMIZED-SELECT: may have to recurse consistently on an array that is only 1 element smaller than previous array (see QuickSort lecture):
  \[ T(n) = T(n - 1) + \theta(n) = \theta(n^2) \]
- Best case for RANDOMIZED-SELECT: consistently recurse on array that was half the size of the previous array:
  \[ T(n) = T\left(\frac{n}{2}\right) + \theta(n) = \theta(n) \]
- Average case: Difficult, but can show \(\theta(n)\) (see book)
- Question: Can we implement a (deterministic) SELECT algorithm in worst-case linear time?
SELECT (A, n, i)

Idea is to force a good split:

1) Divide n elements of A into \( \sim n/5 \) groups of 5 elements
2) Find the median of each of these \( \sim n/5 \) groups
   a. Run insertion-sort on each group of 5
   b. Pick median from 5 elements
3) Use SELECT recursively to find the super median \( x \) of the \( \sim n/5 \) medians (take lower median if even #)
4) PARTITION (deterministically-pivot last element) the n elements around super median \( x \)
   a. Now let \( x \) the \( k^{th} \) element in array after this partitioning
   b. Will be \( k-1 \) elements on low side, \( n - k \) elements on high side of partition

SELECT (A, n, i) continued

5) Three possibilities:
   • If \( i = k \), done, so return \( x \)
   • If \( i < k \), call SELECT recursively to find \( i^{th} \) smallest element on low side
   • If \( i > k \), call select to find \( (i-k)^{th} \) smallest element on high side
SELECT: One iteration

![One iteration on the list (0,1,2,3,...99)]

\[ T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n) \]

which will run in \( O(n) \).

Exercise 9.3-1: In SELECT, divide into groups of 7, Analyze. Still linear? What about groups of 3?

Answer: For any odd group size \( \geq 5 \), the SELECT algorithm is \( O(n) \).
Extension: Finding i Largest Numbers

Problem 9-1: Given a set of $n$ numbers, we wish to find the $i$ largest in sorted order using a comparison-based algorithm. Find the algorithm that implements each of the following methods with the best asymptotic worst-case running time, and analyze the running times of the algorithms in terms of $n$ and $i$.

- Sort the numbers, and list the $i$ largest.
- Build a max-priority queue from the numbers and call EXTRACT-MAX $i$ times.
- Use an order-statistic algorithm to find the $i$th largest number, partition around that number, and sort the $i$ largest numbers.