Medians and Order Statistics

Order Statistics

- *How many comparisons are needed to find the minimum element in a set? The maximum?*
- *Can we find the minimum and maximum with less than twice the cost?*
- Yes:
  - Walk through elements by pairs
    - Compare each element in pair to the other
    - Compare the largest to maximum, smallest to minimum
  - Total cost: 3 comparisons per 2 elements = O(3n/2)

Order Statistics – contd.

- The ith *order statistic* in a set of n elements is the ith smallest element
- That is, ith order statistic of n elements S={a_1, a_2,…, a_n} : ith smallest elements
- The *minimum* is thus the 1st order statistic
- The *maximum* is the nth order statistic
- The *median* is the $n/2$ order statistic
  - If n is even, there are 2 medians

Order Statistics – contd.

- Could calculate order statistics by sorting
  - Time: O(n lg n) with comparison sort
- *How can we better calculate order statistics?*
- *What is the running time?*
- Selection in expected/average linear time

$O(n\lg n)$ Algorithm

- Suppose n elements are sorted by an $O(n\lg n)$ algorithm, e.g., MERGE-SORT
  - Minimum: the first element
  - Maximum: the last element
  - The ith order statistic: the ith element.
  - Median:
    - If n is odd, then $((n+1)/2)$th element.
    - If n is even,
      - then $(\lfloor (n+1)/2 \rfloor)$th element, lower median
      - then $(\lceil (n+1)/2 \rceil)$th element, upper median
- All selections can be done in O(1), so total: $O(n\lg n)$.
- Can we do better?

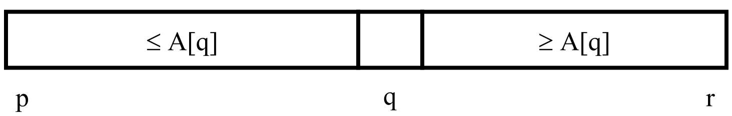
Selection in Expected Linear Time $O(n)$

- Select ith element
- A divide-and-conquer algorithm RANDOMIZED-SELECT
- Similar to quicksort, partition the input array recursively
- Unlike quicksort, which works on both sides of the partition, just work on one side of the partition.
  - **Called** prune-and-search, prune one side, just search the other side).

Randomized Selection

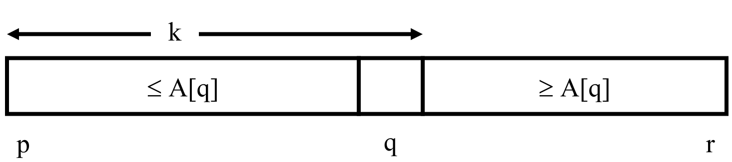
- Key idea: use partition() from quicksort
  - But, only need to examine one subarray
  - This savings shows up in running time: O(n)
- We will again use a slightly different partition than the book:

  q = RandomizedPartition(A, p, r)



Randomized Selection

```
RandomizedSelect(A, p, r, i)
    if (p == r) then return A[p];
    q = RandomizedPartition(A, p, r)
    k = q - p + 1;
    if (i == k) then return A[q];   // not in book
    if (i < k) then
        return RandomizedSelect(A, p, q-1, i);
    else
        return RandomizedSelect(A, q+1, r, i-k);
```



Finding Order Statistics: The Selection Problem

- A more interesting problem is *selection*: finding the *i*th smallest element of a set
- We will show:
  - A practical randomized algorithm with O(n) expected running time
  - A cool algorithm of theoretical interest only with O(n) worst-case running time

Analysis of RANDOMIZED-SELECT

- Worst-case running time $\Theta(n^2)$, why???

It may be unlucky and always partition into A[q], an empty side and a side with remaining elements. So every partitioning of m elements will take $\Theta(m)$ time, and m=n,n-1,…,2.

Thus total is $\Theta(n)+\Theta(n-1)+\ldots+\Theta(2)=\Theta(n(n+1)/2-1)=\Theta(n^2)$. Moreover, no particular input elicits the worst-case behavior, Because of "randomness".

But in average, it is good.

By using probabilistic analysis/random variable, it can be proven that the expected running time is $O(n)$.
Can we do better, such that $O(n)$ in worst case??

Randomized Selection

- Analyzing **`RandomizedSelect()`**
  - Worst case: partition always 0:n-1

    T(n) = T(n-1) + O(n) ***= ???***

    = O(n^2) (arithmetic series)

    - No better than sorting!
  - "Best" case: suppose a 9:1 partition

    T(n) = T(9*n*/10) + O(n) ***= ???***

    = O(n) (Master Theorem, case 3)

    - Better than sorting!
    - *What if this had been a 99:1 split?*

Randomized Selection

- Average case
  - For upper bound, assume *i*th element always falls in larger side of partition:

$$T(n) \le \frac{1}{n}\sum_{k=0}^{n-1} T(\max(k, n-k-1)) + \Theta(n)$$

$$\le \frac{2}{n}\sum_{k=n/2}^{n-1} T(k) + \Theta(n)$$

What happened here?

  - Let's show that T(*n*) = O(*n*) by substitution

Randomized Selection

- Assume T(n) ≤ cn for sufficiently large c:

$$T(n) \le \frac{2}{n}\sum_{k=n/2}^{n-1} T(k) + \Theta(n)$$ ***The recurrence we started with***

$$\le \frac{2}{n}\sum_{k=n/2}^{n-1} ck + \Theta(n)$$ ***Substitute T(n) ≤ cn for T(k)***

$$= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{n/2-1} k\right) + \Theta(n)$$ ***"Split" the recurrence***

$$= \frac{2c}{n}\left(\frac{1}{2}(n-1)n - \frac{1}{2}\left(\frac{n}{2}-1\right)\frac{n}{2}\right) + \Theta(n)$$ ***Expand arithmetic series***

$$= c(n-1) - \frac{c}{2}\left(\frac{n}{2}-1\right) + \Theta(n)$$ ***Multiply it out***

Randomized Selection

- Assume T(n) ≤ cn for sufficiently large c:

$$T(n) \le c(n-1) - \frac{c}{2}\left(\frac{n}{2}-1\right) + \Theta(n)$$ ***The recurrence so far***

$$= cn - c - \frac{cn}{4} + \frac{c}{2} + \Theta(n)$$ ***Multiply it out***

$$= cn - \frac{cn}{4} - \frac{c}{2} + \Theta(n)$$ ***Subtract c/2***

$$= cn - \left(\frac{cn}{4} + \frac{c}{2} - \Theta(n)\right)$$ ***Rearrange the arithmetic***

$$\le cn \quad \text{(if c is big enough)}$$ ***What we set out to prove***