Lecture 5

CS325

Problem: Selection

Problem definition:

Input: A list of numbers A and integer k

Output: the k-th smallest number of A

For simplicity, we can assume all elements are distinct

A generalization of the ***median*** problem, which finds the 50th percentile of S.

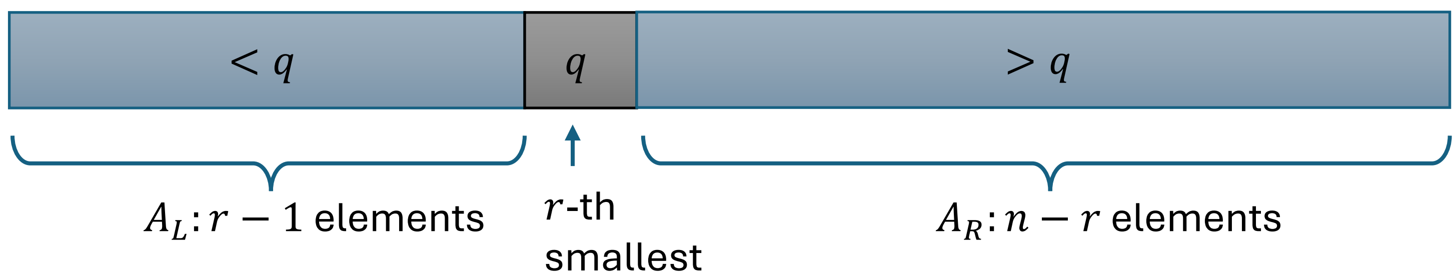
Immediate solution for $O(n \log n)$?

Can we do solve it in $O(n)$?

A recursive solution: rough idea

- Randomly select an element (pivot)

- **Partition** the array according to q



- Recurse to A_L or A_R based on r and k
  - If $k < r$, recurse on $(A_L, ?)$
  - If $k = r$, ?
  - If $k > r$, recurse on $(A_R, ?)$

Pseudocode

```
Partition(A[1,…,n], q)
   i=1, j=n
   for k=1 to n except v:
     if A[k]<A[v]:
       B[i] = A[k], i++
     else:
       B[j] = A[k], j--
   B[i] = A[v]
   copy B to A
   return i
```

Pseudocode

```
Partition(A[1,…,n], q)
  i=1, j=n
  for k=1 to n except v:
    if A[k]<A[v]:
      B[i] = A[k], i++
    else:
      B[j] = A[k], j--
  B[i] = A[v]
  copy B to A
  return i
```

```
Select(A[1,…,n], k)
  v = a random # in {1,…,n}
  r = partition(A, v)
  if r=k:
     return A[r]
  if r<k:
     Select(A[1,…,r-1],k)
  if r>k:
     Select(A[r+1,…,n],k-r)
```

Runtime analysis

$$T(n) = \mathrm{O}(\mathrm{n}) + \max\{1, T(r-1), T(n-r)\}$$

$$= O(n) + \max\{T(r-1), T(n-r)\}$$

where r is a random number in {1,...,n}

Good case: $r = \frac{n}{2}$

Bed case: $r = 1$ or $n - 1$

Runtime analysis

$$T(n) = \mathrm{O(n)} + \max\{1, T(r-1), T(n-r)\}$$

$$= O(n) + \max\{T(r-1), T(n-r)\}$$

where r is a random number in {1,…,n}

Good case: $r = \frac{n}{2}$

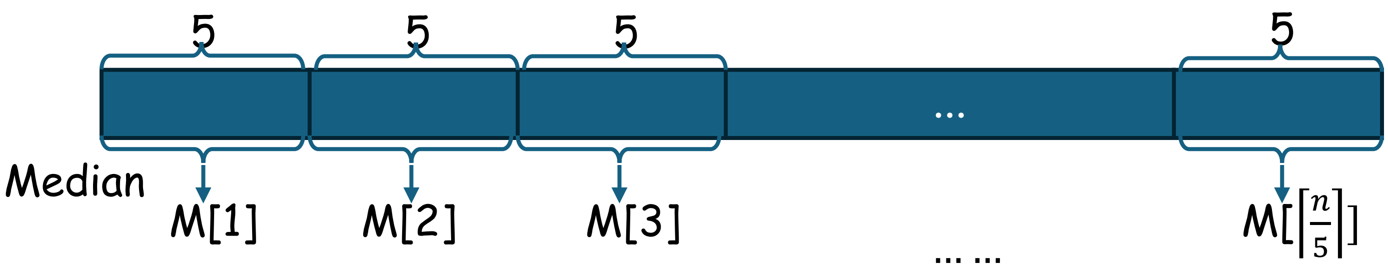
$$T(n) = O(n)$$

Bed case: $r = 1$ or $n - 1$

$$T(n) = O(n^2)$$

Smart* selection of pivot

- Break A into chunks of fives and find the median of each chunk



- Use the median of M to be the pivot

- **Claim:**

Median of M is larger than $\frac{3}{10}$ of A and smaller than $\frac{3}{10}$ of A

```
Select(A, k)
1.   for i=1,…,n/5
2.       M[i] = median (A[5(i-1)+1, 5i])
3.   MM = Select(M, n/10)
4.   v = index of MM in A
5.   r = partition(A, v)
6.   if k=r: return A[r]
7.   if k<r: Select(A[1,…,r-1], k)
8.   if k>r: Select(A[r+1,…,n], k-r)
```

$$T(n) \leq O(n) + T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right)$$
$$\Rightarrow T(n) = O(n)$$

Practical considerations

- The “median of median” method has a better theoretical runtime – worst case $O(n)$
- Practically it is not really used
  - The overhead of identifying the median of median is generally too big for it to be practically useful
  - The original randomized algorithm has average runtime $O(n)$ --- see DPV discussion on this

- Similar idea is behind the popular quicksort algorithm
- Partition can be implemented in place (without requiring additional memory allocation)
  - See figure 1.8 here for the algorithm https://jeffe.cs.illinois.edu/teaching/algorithms/book/01-recursion.pdf#page=9.10

Example Problem: D & C

Input: an array A of sorted integers that have been shifted.

Goal: find the largest element in A

Example:

(40, 57, 89, 2, 8, 25, 30)

shifted 3 positions

Brute force?

Can we do better?

Goal: O(log n) run time

Q: how can we achieve O(log n) time?

A: Each recursion

- do some constant time operation
- shrink the input size by a constant factor. e.g. to n/2 like binary search

Rough idea:

(40, 57, 89, 2, 8, 25, 30)

- Identify the middle element,
- Do constant time operation on it
- Eliminate half of the array