SELECTION

Thursday, October 11, 2018 5:09 PM

IDEA: always use median as pivot $\Rightarrow$ worst-case QS becomes $\Theta(n \lg n)$
IF we can find median in $\Theta(n)$

SELECTION

INPUT: unsorted array $A[lo..hi]$, integer k, $1 \le k \le n = hi - lo + 1$

OUTPUT: the element in $A[lo..hi]$ with rank k, i.e. the k^{th} smallest element.

In particular, select $(A[lo..hi], \frac{lo+hi}{2})$ → rank of the median (left) yields the median

Brute: find the element with rank k, i.e. it is bigger than $k-1$ other elements exactly

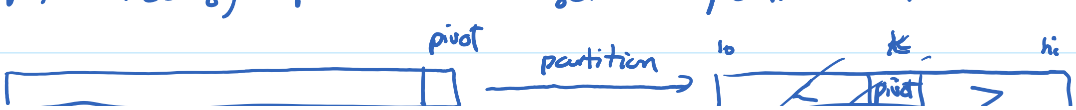
```
brute-select (A[lo..hi], k)
{
    for (i = lo; i ≤ hi; ++i)
    {
        count = 0;
        for (j = lo; j ≤ hi; ++j)
            if A[j] < A[i]
                ++count;
        if (count == k-1)
            return A[i];
    }
}
```

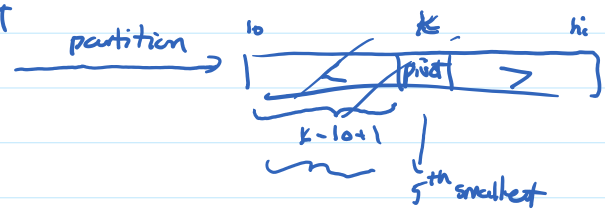
Running time: $\sum_{i=lo}^{hi} \sum_{j=lo}^{hi} (1) = n^2$ in the worst case.

```
transform-select (A[lo..hi], k)
{
    merge_sort (A[lo..hi]);
    return A[lo+k-1];
}
```

Running time: $\Theta(n \lg n)$

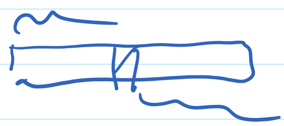
Divide-conquer: modify quicksort to get a "partial" sort





rank of pivot : $k - lo + 1$

```
quickselect (A[lo..hi], k)
{
    if (lo == hi)
        return A[lo];
    pivot_pos = partition (A[lo..hi]);
    pivot_rank = pivot_pos - lo + 1;
    switch (comp(pivot_rank, k))
    {
      case 0:  return A[pivot_pos];
      case +:  return quick-select (A[lo..pivot_pos-1], k);
      case -:  return quick-select (A[pivot_pos+1..hi], k - pivot_rank);
    }
}
```



$C(1) = 0$

$C(n) = n-1 + C(\ n-1\ )\ ,\ n > 1.$

$$C(n) = n-1 + n-2 + C(n-2)$$
$$= n-1 + n-2 + \cdots\cdots + 1 + C(1)$$
$$= \frac{(n-1)n}{2} \in \Theta(n^2)$$

Idea: handpick pivot to avoid worst-case behavior

Ideally, we pick the median. But this is the selection problem.

<u>KEY INSIGHT</u>: no need to use exact middle value (median)

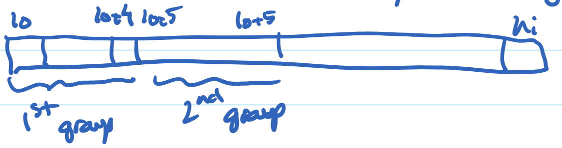
ANY value whose rank is in the middle third will do

sorted array

2 issues: Ⓐ how to find a value whose rank is in the middle third?

Ⓑ why using this value as the pivot yields worst-case linear time?

(A) 1) partition unsorted array into groups of 5



2) insertion sort each group INDEPENDENTLY

n = array size

g = # of groups $= \lfloor \frac{n}{5} \rfloor$

each group requires $10 = \frac{5 \times 4}{2}$ comparisons for insertion sort

In total: $10 \times \lfloor \frac{n}{5} \rfloor \le 2n$

$\underbrace{a_1 < a_2 < a_3 < a_4 < a_5}_{1^{st}\text{ group}} \quad \underbrace{b_1 < b_2 < b_3 < b_4 < b_5}_{2^{nd}\text{ group}} \quad \underbrace{c_1 < c_2 < c_3 < c_4 < c_5}_{3^{rd}} \quad \ldots$

$M[\,] = a_3 \;\; b_3 \;\; c_3 \;\; \ldots\ldots$

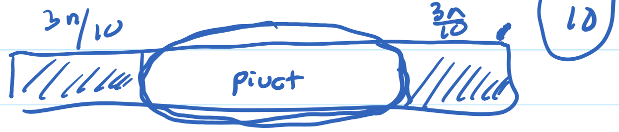
CLAIM: the median of these medians has rank in the middle third obtained by a recursive call

```
linear-select (A[lo..hi], k)
{
    n = hi-lo+1;
    if (n <5)
    {
        insertion_sort (A[lo..hi]);
        return A[lo+k-1];
    }
    g = n/5;
    for (i = 0; i <g; ++i)
    {
        insertion_sort (A[lo+ 5*i .. lo+5*i+4]);
        M[i] = A[lo+ 5*i +2];
    }
    med = linear_select (M[0..g-1], g/2+1); // median of medians
    for (i= lo; i <= hi; ++i)
        if (A[i] == med)
```

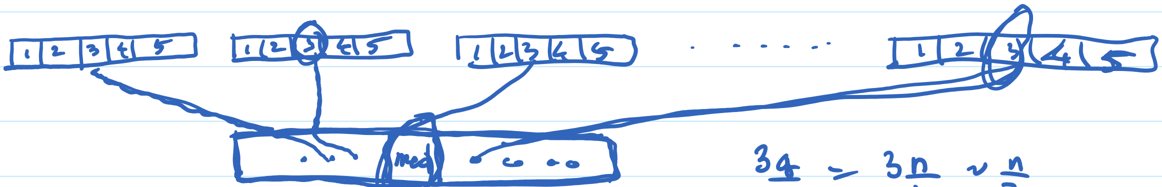
```
    { swap(A[i], A[hi]); break }

  pivot_pos = partition(A[lo..hi])
  pivot_rank = pivot_pos - lo + 1;
  switch ( comp(pivot_rank, k) )
  {
     case 0 : return A[pivot_pos];
     case + : return linear_select(A[lo..pivot_pos-1], k);
     case - : return linear_select(A[pivot_pos+1..hi], k-pivot_rank);
  }
}
```

CLAIM: the handpicked pivot (median of medians) is bigger than at least $\frac{3n}{10}$ elements and smaller than at least $\frac{3n}{10}$ elements



Proof:



$\frac{3g}{2} = \frac{3n}{10} \sim \frac{n}{3}$

sorted M array

$\frac{g}{2} = \frac{n}{10} = \frac{g}{2}$

CLAIM: The worst-case running time of linear select is $\Theta(n)$

Proof: We count # of array element comparisons

$$C(4) = 6$$
$$C(n) = 10\left(\lfloor \tfrac{n}{5} \rfloor\right) + C\left(\lfloor \tfrac{n}{5} \rfloor\right) + n + C\left(\lceil \tfrac{7n}{10} \rceil\right)$$

Assume n is a power of 10 to remove $\lfloor\ \rfloor$, $\lceil\ \rceil$

$$C(n) = C\left(\frac{2n}{10}\right) + C\left(\frac{7n}{10}\right) + 3n$$

Easy proof by induction that $C(n) \le 40n$ when $n \ge 289$.

NOTE: linear-select runs in worst-time linear time, but because the constant is

ing, it is not competitive against quick select on RANDOM inputs.
In fact, the average running time of quick select is

$$2n - 2H(n)$$

CLAIM: Average running time of QUICKSELECT is $2n - 2H(n)$

Let $S(n)$ = average running time of QuickSelect on arrays of size n.

$$S(1) = 0$$

$$S(n) = n-1 + \frac{1}{n}\sum_{i=0}^{n-1} S(i)$$ // assume that recursion size i is equally likely for $0 \le i \le n-1$

$$nS(n) = n(n-1) + \sum_{i=0}^{n-1} S(i)$$ true for $n > 1$

$$(n-1)S(n-1) = (n-1)(n-2) + \sum_{i=0}^{n-2} S(i)$$

$$nS(n) - (n-1)S(n-1) = 2(n-1) + S(n-1)$$

$$nS(n) = 2(n-1) + nS(n-1)$$

$$S(n) = \frac{2(n-1)}{n} + S(n-1)$$

$$\boxed{S(n) = 2 - \frac{2}{n} + S(n-1)}$$

$$= 2 - \frac{2}{n} + 2 - \frac{2}{n-1} + S(n-2)$$

$$= 2.2 - 2\left(\frac{1}{n} + \frac{1}{n-1}\right) + S(n-2)$$

$$= \dots$$

$$= 2(n-1) - 2\left(\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{2}\right) + \underbrace{S(1)}_{=0}$$

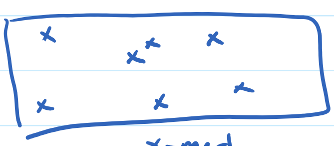
$$= 2(n-1) - 2(H(n) - 1) + 0$$

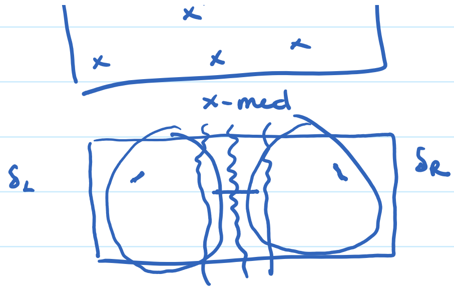
$$= 2n - 2 - 2H(n) + 2$$

$$= 2n - 2H(n)$$

OTHER DIVIDE-CONQUER ALGS

closest pair: given n points on the plane, smallest distance between any 2 of the given points





$\delta = \min(\delta_L, \delta_R)$

Fast Fourier Transform

any 2 of the given points