Lecture 20

COL 351: Analysis and Design of Algorithms

Lecture 20

Quick Sort

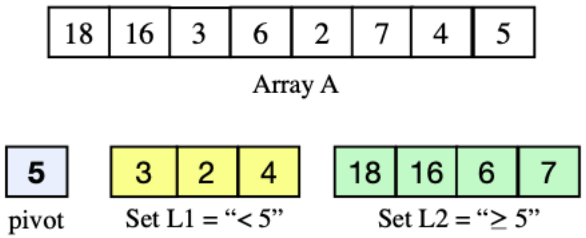
(Divide and Conquer strategy)

(Divide and Conquer strategy)

Divide and Conquer

1. **Divide the main problem into smaller subproblems.**
2. **Solve the smaller sub-problems recursively**
3. **Combine**

Quick Sort



1. Divide into smaller subproblems

5		2	3	4		6	7	16	18

pivot Sorted L1 Sorted L2

2. Solve the smaller sub-problems recursively

2	3	4	5	6	7	16	18

3. Combine

Deterministic Quick Sort

```
DetQuickSort(L)

    x = Median of list L;                    /* pivot is median*/

    Initialise L1 and L2 to be empty lists;

    For each (y ∈ L\x):
        If (y ⩽ x) : L1.append(x);
        If (y > x) : L2.append(x);

    Return DetQuickSort(L1) ∘ x ∘ DetQuickSort(L2);
```

How to compute median?

$\boxed{\text{Median } (A) - \left(\frac{n}{2}\right)^{th} \text{ position element in sorted } A.}$ $\Rightarrow |L1|, |L2| \leq n/2$

New Problem: Computing kth-smallest element

For Median

$i = 1$

$j = n = |A|$

$k = \frac{n}{2} \,/\, \frac{n+1}{2}$

Given: A sub-array $A[i, j]$ and an integer $k \in [i, j]$.

Find: The element at $A[k]$ if we sort $A[i, j]$.

Example:

A = [77, 33, 88, 66, 44, 11, 22, 55]

If $i = 1$, $j = 8$, $k = 2$ then we need to return 22.

Sorted(A) → 11 $\underline{22}$ 33 44 55 66 77 88

k=2

Computing kth-smallest element

Given: A sub-array $A[i, j]$ and an integer $k \in [i, j]$.

Find: The element at $A[k]$ if we sort $A[i, j]$.

Algorithm Sketch:

1. Find an appropriate element $x \in A[i, j]$
2. $pos =$ position of x in Sorted $A[i, j]$.
3. Re-arrange elements in $A[i, j]$ so that - $A[i], \ldots, A[pos-1] \le x = A[pos] \le A[pos+1], \ldots, A[j]$.
4. If $(pos = k)$ then return x.
5. If $(k < pos)$ then Search in sub-array $A[i, pos-1]$
6. If $(pos < k)$ then Search in sub-array $A[pos+1, j]$

Example:

A = [77, 33, 88, 66, 44, 11, 22, 55], $k = 2$, search in A[1, 8]

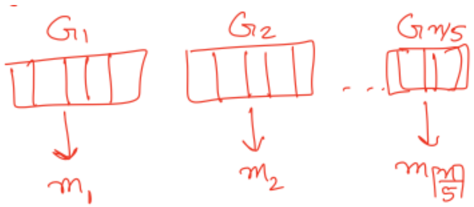
roughly balanced

$x = 55$

A<55 A>55

$\underbrace{33, 44, 11, 22}_{\text{New search range} \uparrow}$ $\underbrace{55}_{5^{th} \text{ pos}}$ $\underbrace{77, 88, 66}$

Computing kth-smallest element



Order$(A, i, j, order = k)$

1a. Divide $A[i, j]$ into $\lceil n/5 \rceil$ groups each of size 5.

1b. U = Array containing medians of all $\lceil n/5 \rceil$ groups.

suppose $n_0 = |A[i,j]|$

1c. $x = Order(U, 1, \lceil n/5 \rceil, order = \lceil n/10 \rceil)$. $|U|/2$

2. $pos =$ position of x if we sort $A[i, j]$.

3. Re-arrange elements in $A[i, j]$ so that
   $A[i], \ldots, A[pos-1] \leq x = A[pos] \leq A[pos+1], \ldots, A[j]$.

4. If $(pos = k)$ return x.

5. If $(pos > k)$ return $Order(A, i, pos-1, order = k)$.

6. If $(pos < k)$ return $Order(A, pos+1, j, order = k)$.

Time Complexity

Step 1:

$O(n_0) + T(\lceil n_0/5 \rceil)$

Step 2-4:

$O(n_0)$

Step 5-6:

CLAIM: $T(\frac{7}{10} n_0)$

Computing kth-smallest element

$Order(A, i, j, order = k)$

1a. Divide $A[i, j]$ into $\lceil n/5 \rceil$ groups each of size 5.

1b. $U =$ Array containing medians of all $\lceil n/5 \rceil$ groups.

1c. $x = Order(U, 1, \lceil n/5 \rceil, order = \lceil n/10 \rceil)$. $|U| = \lceil \frac{n_0}{5} \rceil$

2. $pos =$ position of x if we sort $A[i, j]$.

3. Re-arrange elements in $A[i, j]$ so that
   $A[i], \ldots, A[pos-1] \leq x = A[pos] \leq A[pos+1], \ldots, A[j]$.

4. If $(pos = k)$ return x. $\leq \frac{7}{10} n_0$

5. If $(pos > k)$ return $Order(A, \boxed{i, pos-1,}\ order = k)$.

6. If $(pos < k)$ return $Order(A, \boxed{pos+1, j,}\ order = k)$.

Let $n_0 = j - i + 1$ $= |A[i,j]|$

Steps 1: $O(n_0) + T(n_0/5)$

Steps 2, 3: $O(n_0)$

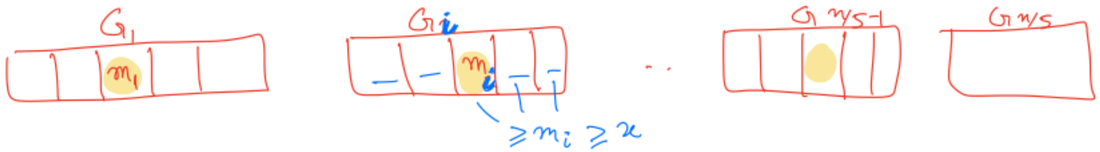
Steps 5, 6: ? $T(\frac{7 n_0}{10})$

Note: $n_0/10$ elements in U are $\geq x$.

So at least $3(n_0/10)$ elements in $A[i, j]$ are $\geq x$.

$\Rightarrow |A[i, pos-1]| \leq 7(n_0/10)$

Similarly, $|A[pos+1, j]| \leq 7(n_0/10)$

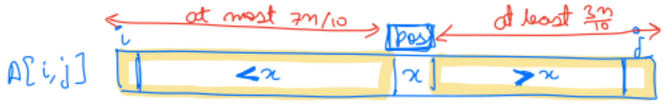


In "U" x is median

$\rightarrow$ $|U|$ elements, i.e. $\frac{n}{10}$ elements in U are $\geq x$

If $\underline{m_i \geq x}$ $\Rightarrow$ 3 elements in G_i are $\geq x$

$\underbrace{\# \text{ of } m_i\text{'s} \geq x}$ is $\frac{n}{10}$ $\Rightarrow$ at least $\frac{3n}{10}$ elements in $A[i,j]$ $\geq x$



Computing kth-smallest element

Order$(A, i, j, order = k)$

1a. Divide $A[i, j]$ into $\lceil n/5 \rceil$ groups each of size 5.

1b. U = Array containing medians of all $\lceil n/5 \rceil$ groups.

1c. $x = Order(U, 1, \lceil n/5 \rceil, order = \lceil n/10 \rceil)$.

2. pos = position of x if we sort $A[i, j]$.

3. Re-arrange elements in $A[i, j]$ so that
   $A[i], \ldots, A[pos-1] \leq x = A[pos] \leq A[pos+1], \ldots, A[j]$.

4. If $(pos = k)$ return x.

5. If $(pos > k)$ return $Order(A, i, pos-1, order = k)$.

6. If $(pos < k)$ return $Order(A, pos+1, j, order = k)$.

Let $n_0 = j - i + 1$

Steps 1: $O(n_0) + T(n_0/5)$

Steps 2, 3: $O(n_0)$

Steps 5, 6: $T(7n_0/10)$

Time complexity follows the relation

$T(n) = \underline{T(n/5)} + \underline{T(7n/10)} + \underline{O(n)}$ cn

Assume, $\forall \alpha \leq n$, $T(\alpha) \leq 10 \cdot c \cdot \alpha$

$T(n/5) \leq 2cn$

$T(\frac{7n}{10}) \leq 7cn$

cn

Computing kth-smallest element

Lemma: The k^{th} smallest element of a list L of size n is computable in $O(n)$ time.

Corollary: The Median of a list L of size n is computable in $O(n)$ time.

Extra Reading: Fast Deterministic Median (http://erdani.com/research/sea2017.pdf)

Deterministic Quick Sort

DetQuickSort(L)

x = Median of list L;

Initialise L1 and L2 to be empty lists;

For each ($y \in L \backslash x$):
  If ($y \leqslant x$) : L1.append(x);
  If ($y > x$) : L2.append(x);

Return $\underbrace{\textit{DetQuickSort}(L1)}_{T(n/2)} \circ x \circ \underbrace{\textit{DetQuickSort}(L2)}_{T(n/2)}$;

Time complexity follows the relation

$T(n) = 2T(n/2) + O(n)$

On solving we get

$T(n) = O(n \log n)$

Constants in the big Oh expression are large, so Randomised Quick Sort is more practical!

$$T(n) = 2T(n/2) + cn$$

$$= 2\left(2T\left(\frac{n}{4}\right) + \frac{cn}{2}\right) + cn$$

$$= 4\,T\left(\frac{n}{4}\right) + 2cn$$

$$= 8\,T\left(\frac{n}{8}\right) + 3cn$$

$$\vdots$$

$$i = \log_2 n \qquad = \underbrace{2^i\,T\left(\frac{n}{2^i}\right)}_{} + \underbrace{icn}_{}$$

$$= O(n) + O(n\log n) = O(n\log n)$$

Randomized Quick Sort

RandQuickSort(L)

> x = Uniformly Random element chosen from list L;
>
> Initialise L1 and L2 to be empty lists;
>
> For each ($y \in L \backslash x$):
> If ($y \leqslant x$) : L1.append(x);
> If ($y > x$) : L2.append(x);
>
> Return *RandQuickSort*(L1) ∘ x ∘ *RandQuickSort*(L2);

Take pivot to be any random element.

Expected time complexity $- O(n\log n)$

99.999% times randomized quick sort behaves better than det quick sort