Lecture Note 2

CSCI 6470 Algorithms (Fall 2024)

Liming Cai

School of Computing UGA

September 5, 2024

Chapter 2 Power of Divide-and-Conquer

1. Divide-and-conquer approach
2. More on solving recurrence relations
3. Quick sort and average case time complexity
4. Algorithms for order statistics

1. Divide-and-conquer approach

- divide-and-conquer is a top-down design approach;
- it breaks the task into several subtasks;
- subtasks are usually solved recursively;

e.g., merge sort, binary search, quick sort, time complexities tend to look like:

$$T(n) = \sum_{i=1}^{k} T(\beta_i n) + B(n)$$

where the task of size n is broken down into k subtasks of the same nature, each with size $\alpha_i n$ for percentage $\beta_i < 1$, $i = 1, 2, \ldots, k$, subject to

$$\sum_{i=1}^{k} \beta_i = 1$$

Some algorithms have $<$ instead of $=$.

1. Divide-and-conquer approach

Merge Sort

```
function MergeSort(L, low, high);
if low < high    \\ at least 2 elements
    mid = floor((low + high)/2);
    MergeSort(L, low, mid);
    MergeSort(L, mid+1, high);
    MergeTwo(L, low, mid, high);
    return;
```

Its time complexity has parameters $\beta_1 = \beta_2 = \frac{1}{2}$, $B(n) = O(n)$. That is

$$T(n) = \begin{cases} a & n \leq 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + bn & n \geq 2 \end{cases}$$

1. Divide-and-conquer approach

Multiplication of two n-bits numbers:

- a recursive strategy:

$$Prod(x,y) = \begin{cases} 2 \times Prod(x, \frac{y}{2}) & \text{if } y \text{ is even} \\ x + 2 \times Prod(x, \lfloor \frac{y}{2} \rfloor) & \text{if } y \text{ is odd} \end{cases}$$

e.g.,

$$9 \times 11 = 9 + 2 \times (9 \times \lfloor \frac{11}{2} \rfloor)$$

Let $T(n)$ be the time complexity for $Prod(x,y)$ when $|y| = n$.

$$T(n) = \begin{cases} a & n = 1 \\ T(n-1) + bn & n \geq 2 \end{cases}$$

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

1. Divide-and-conquer approach

Multiplication of two n-bits numbers:

- a divide-and-conquer strategy

  (1) split x and y into high and low segments, each with $\frac{n}{2}$ bits;
  (2) $x \times y$ uses 4 $\times$'s on segments plus some additions;

$$\begin{aligned} xy &= (x_h 2^{\frac{n}{2}} + x_l)(y_h 2^{\frac{n}{2}} + y_l) \\ &= x_h y_h 2^n + (x_h y_l + x_l y_h) 2^{\frac{n}{2}} + x_l y_l \end{aligned}$$

  e.g., for decimal numbers (based 10 instead)

  Time complexity:

$$T(n) = \begin{cases} a & n = 1 \\ 4T(\frac{n}{2}) + bn & n \geq 2 \longleftarrow \text{why?} \end{cases}$$

  $T(n) = O(n^2)$ can you prove?

1. Divide-and-conquer approach

Multiplication of two n-bits numbers:

- a better solution

$$
\begin{aligned}
xy &= (x_h 2^{\frac{n}{2}} + x_l)(y_h 2^{\frac{n}{2}} + y_l) \\
&= x_h y_h 2^n + (x_h y_l + x_l y_h) 2^{\frac{n}{2}} + x_l y_l
\end{aligned}
$$

where with 1 '$\times$' operation

$$(x_h y_h + x_l y_h) = (x_h + x_l)(y_h + y_l) - x_h y_h - x_l y_l$$

$T(n) = 3T(n/2) + O(n)$ leading to

$T(n) = O(n^{1.6})$ can you prove?

1. Divide-and-conquer approach

Matrix multiplication

- for 2×2 matrices: $A_{(2\times 2)} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

$$A_{(2\times 2)} \times B_{(2\times 2)} = C_{(2\times 2)}$$

conventional way: 8 scalar multiplications needed

- for $n \times n$ matrices: $A_{n\times n} \times B_{n\times n} = C_{n\times n}$

$$A_{(n\times n)} = \begin{bmatrix} X_{(\frac{n}{2}\times\frac{n}{2})} & Y_{(\frac{n}{2}\times\frac{n}{2})} \\ Z_{(\frac{n}{2}\times\frac{n}{2})} & W_{(\frac{n}{2}\times\frac{n}{2})} \end{bmatrix}$$

conventional way: 8 $\mathrm{dim}_{(\frac{n}{2}\times\frac{n}{2})}$-matrix multiplications needed
$T(n) = 8T(n/2) + O(n^2)$

then $T(n) = O(n^3)$, prove by unfolding or induction

1. Divide-and-conquer approach

Matrix multiplication (a better solution)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} ax + bz & ay + bw \\ cx + dz & cy + dw \end{bmatrix}$$

more clever algebra, with

$$\begin{cases} s_1 = a(y - w) & s_5 = (a + d)(x + w) \\ s_2 = (a + b)w & s_6 = (b - d)(z + w) \\ s_3 = (c + d)x & s_7 = (a - c)(x + y) \\ s_4 = d(z - x) & \end{cases}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} s_4 + s_5 + s_6 - s_2 & s_1 + s_2 \\ s_3 + s_4 & s_1 + s_5 - s_3 - s_7 \end{bmatrix}$$

7 multiplications and 18 additions/subtractions !

$T(n) = 7T(n/2) + O(n^2)$, leading to $T(n) = O(n^{2.81})$.

prove by unfolding or induction

2. More on solving recurrence relations

We can prove: for Merge Sort, $T(n) = O(n \log_2 n)$.

Actually, given the recurrence relation for time function

$$T(n) = \sum_{i=1}^{k} T(\beta_i n) + B(n)$$

if $B(n)$ is restricted to $O(n)$, we can prove that $T(n) = O(n \log_2 n)$.

2. More on solving recurrence relations

Theorem. *Assume time function $T(n)$ of some algorithm* has the following recurrence, for fixed number a, b, k, α and β, where $\alpha + \beta = 1$,

$$T(n) = \begin{cases} a & n \leq k \\ T(\alpha n) + T(\beta n) + bn & n > k \end{cases}$$

Then $T(n) = O(n \log_2 n)$, actually, $T(n) = \Theta(n \log_2 n)$.

Proof. Use strong math induction to prove the following claim instead:

There exist $c > 0$ and n_0 such that $T(n) \leq cn \log_2 n$ for all $n \geq n_0$.

2. More on solving recurrence relations

(cont') Proof of the above theorem with strong math induction:

basis: for $n \leq k$, $T(n) = a$. To allow $T(n) = a \leq cn\log_2 n$, it suffices to choose ... what happens here? how to fix it?

assumption: $T(\alpha n) \leq c(\alpha n)\log_2(\alpha n)$ and $T(\beta n) \leq c(\beta n)\log_2(\beta n)$

induction:

$$\begin{aligned}
T(n) &= T(\alpha n) + T(\beta n) + bn \\
&\leq c(\alpha n)\log_2(\alpha n) + c(\beta n)\log_2(\beta n) + bn \\
&= cn\log_2 n + cn\alpha\log_2\alpha + c\beta n\log_2\beta + bn \\
&= cn\log_2 n - n\big(c\alpha\log_2\frac{1}{\alpha} + c\beta\log_2\frac{1}{\beta} - b\big) \\
&\leq cn\log_2 n \quad \text{when } c \geq \frac{b}{(\alpha\log_2\frac{1}{\alpha} + \beta\log_2\frac{1}{\beta})}
\end{aligned}$$

Taken-home exercises II(A)

1. Consider the following recurrences for $T(n)$, which all have $T(1) = a$ as the base case, and for $n \geq 2$,

   (1) $T(n) = 2T(n/2) + bn$;

   (2) $T(n) = 3T(n/2) + bn$;

   (3) $T(n) = 4T(n/2) + bn$;

   Prove that

   (1) $T(n) = O(n \log_2 n)$, (2) $T(n) = O(n^{1.6})$, and (3) $T(n) = O(n^2)$

   respectively, using the unfolding method ("sum of (in)equalities").

2. Using the strong math induction to prove the above big-O results for Question 1.

3. Quick sort algorithms

Quick Sort Algorithm

Idea:

- select a pivot element e from the input list L;
- partition list L into two sublists L_h and L_l such that
  $\forall x \in L_h, x > e$
  $\forall x \in L_l, x \leq e$
- recursively sort the two sublists L_h and L_l, separately;

```
function quicksort(L, low, high);
if (low < high)
   k = partition (L, low, high);
   quicksort(L, low, k-1);
   quicksort(L, k+1, high);
return;
```

3. Quick sort algorithms

How does partition work?

```
function partition(L, p, r);

1. e = L[r];
2. i = p-1;
3. for j = p to r-1
4.   if L[j] <= e
5.     i = i + 1;
6.     exchange (L[i], L[j]);
7. exchange (L[i+1], L[r]);
8. return i+1
```

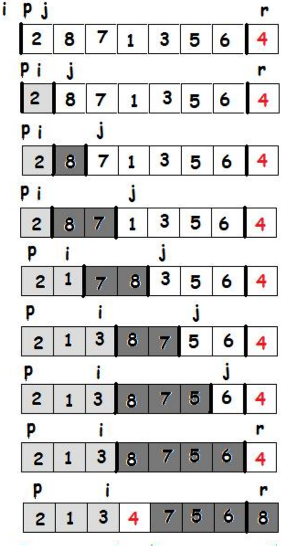
Single pass, dynamically 3 regions:

L[p..i]:

L[i+1..j-1]:

L[j..r-1]:

before pivot is in position $\longrightarrow$



3. Quick sort algorithms

Time complexity, let $n = \texttt{high} - \texttt{low} + 1$;

$$T(n) = \begin{cases} a & n \leq 1 \\ T(|L_h|) + T(|L_l|) + T_P(n) & n \geq 2 \end{cases}$$

where $T_P(n)$ is the time to partition (and to find a pivot from) a list of length n. $T_P(n) = O(n)$

But how big are $|L_h|$ and $|L_l|$?

- note:$|L_h| + |L_l| = n - 1$;
- there is no guarantee that either is a fraction of n in the worst case;
- however, with high probability $|L_h|$ and $|L_l|$ are fractions of n,

  e.g., with 80% chance, $\frac{n}{10} \leq |L_h| \leq \frac{9n}{10}$,

    (1) why is this true?

    (2) why is this fact useful?

3. Quick sort algorithms

Worst case time complexity of Quick Sort

- recursive case: $T(n) = T(|L_h|) + T(|L_l|) + bn$
- a bad case is when L is sorted or reversely sorted, yielding

$$T(n) = \begin{cases} a & n = 1 \\ T(n-1) + bn & n \geq 2 \end{cases}$$

- $T(n) = O(n^2)$ or should we say $\exists c > 0, T(n) \geq cn^2$ instead?

3. Quick sort algorithms

Average case time complexity of Quick Sort

- Assumption: n elements on input list L are in the uniform distribution
- the pivot (say, last element) has 80% chance to partition the list into two sublists of ratio $\frac{1}{10}n : \frac{9}{10}n$ (or better)
- this has high chance to lead to time complexity $T(n) = O(n \log_2 n)$
- this is the **average case time complexity** because there are many cases of pivots

3. Quick sort algorithms

- if the elements in the input list L are not in in the uniform distribution, the quick sort algorithm can randomly shuffle them to generate such a distribution
- this leads to the following `randomized-quicksort` algorithm

```
function randomized-quicksort(L, low, high);
1. if low < high
2.     randomly choose index m between low and high;
3.     swap(L[m], L[high]);
4.     k = partition(L, low, high);
5.     randomized-quicksort(L, low, k-1);
6.     randomized-quicksort(L, k+1, high);
```

3. Quick sort algorithms

Average case time: $\hat{T}(n) =$ (sum of times incurred by different pivots)$/n$

$$\hat{T}(n) = \frac{1}{n}\big(bn + \hat{T}(n-1) + \hat{T}(0) + \\ bn + \hat{T}(n-2) + \hat{T}(1) + \\ \cdots\cdots \\ bn + \hat{T}(1) + \hat{T}(n-2) + \\ bn + \hat{T}(0) + \hat{T}(n-1)\big)$$

Equivalently, calculated as expected time:

$$\hat{T}(n) = \sum_{i=1}^{n} P(L[i] \text{ is pivot}) \times \text{ sorting time incurred by } L[i]$$

$$\hat{T}(n) = \begin{cases} a & n \leq 1 \\ bn + \frac{1}{n}\sum_{i=1}^{n}\big(\hat{T}(i-1) + \hat{T}(n-i)\big) & n \geq 2 \end{cases}$$

$\hat{T}(n) = O(n \log_2 n)$ provable with strong math induction!

3. Quick sort algorithms

Theorem: The average case time complexity of random quick sort algorithm is $O(n \log_2 n)$ on input list of n elements.

Proof outline (proof-by-induction):
It is equivalent to proving $\exists c > 0, n_0 > 0, T(n) \leq cn \log_2 n$, for $n \geq n_0$.

- proof for base case $n =$?;
- assumption: for all $i = 0, 1, 2, \ldots, k-1,\ T(i) \leq ci \log_2 i$;
- induction:

$$T(k) = bk + \frac{1}{k} \sum_{i=0}^{k-1} \Big(T(i) + T(k-i-1) \Big)$$

$$= bk + \frac{2}{k} \sum_{i=0}^{k-1} T(i)$$

$$\leq bk + \frac{2}{k} \sum_{i=0}^{k-1} c \times i \times \log_2 i \quad \text{by assumptions}$$

3. Quick sort algorithms

$$
\begin{aligned}
T(k) &= bk + \frac{2}{k}\Big(\sum_{i=0}^{k/2} c \times i \times \log_2 i + \sum_{i=k/2+1}^{k-1} c \times i \times \log_2 i\Big) \\
&\leq bk + \frac{2}{k}\Big(\sum_{i=0}^{k/2} c \times i \times \log_2 \frac{k}{2} + \sum_{i=k/2+1}^{k-1} c \times i \times \log_2 k\Big) \\
&\leq bk + \frac{2}{k}\Big(\sum_{i=0}^{k/2} c \times i \times (\log_2 k - \log_2 2) + \sum_{i=k/2+1}^{k-1} c \times i \times \log_2 k\Big) \\
&= bk + \frac{2}{k}\sum_{i=0}^{k-1} c \times i \times \log_2 k - \frac{2}{k}\sum_{i=0}^{k/2} c \times i \\
&= bk + c\frac{2}{k}\,\frac{k-1}{2}(k-1+1)\log_2 k - c\frac{2}{k}\,\frac{k/2}{2}(k/2+1) \\
&= bk + c(k-1)\log_2 k - c\frac{k/2+1}{2} \\
&= ck\log_2 k + bk - c\log_2 k - c\frac{k}{4} - c/2 \\
&\leq ck\log_2 k - (ck/4 - bk) \\
&\leq ck\log_2 k \quad \text{when } c \geq 4b \text{ and } k \geq 1
\end{aligned}
$$

3. Quick sort algorithms

An alternative way to analyze the average case time complexity for quick sort:

- calculate the average number of comparisons used; why enough?
- assume $e_1, \ldots, e_n$ are elements in the order after they are sorted;
- Let random variable $X_{i,j}$ = number of times e_i and e_j are compared;
- fact 1: e_i and e_j are compared only when either is a pivot;
- fact 2: $X_{i,j} = 0$ or 1; why?

3. Quick sort algorithms

- total number of comparisons $\sum_{i<j} X_{i,j}$;
- average number of comparison $E[\sum_{i<j} X_{i,j}] = \sum_{i<j} E[X_{i,j}]$
- $E[X_{i,j}] = P\big(X_{i,j} = 1\big) \times 1 + P\big(X_{i,j} = 0\big) \times 0 = P\big(X_{i,j} = 1\big)$
- $P\big(X_{i,j} = 1\big) = P\big(e_i \text{ or } e_j \text{ is pivot}\big) = \frac{2}{|L_{i,j}|}$, $e_i, e_j \in$ sublist $L_{i,j}$; why?
- $|L_{i,j}| \geq j - i + 1$; so $P\big(X_{i,j} = 1\big) = \frac{2}{|L_{i,j}|} \leq \frac{2}{j-i+1}$;

$$
\begin{aligned}
E\Big[\sum_{i<j} X_{i,j}\Big] &\leq 2\sum_{i<j} \frac{1}{j-i+1} \\
&= 2\sum_{i=1}^{n-1}\sum_{j=2}^{n} \frac{1}{j-i+1}; \\
&\leq 2n\sum_{i=1}^{1}\sum_{j=2}^{n} \frac{1}{j-1+1} \\
&=\leq 2n\sum_{j=2}^{n} \frac{1}{j} = O(n\log_2 n)
\end{aligned}
$$

4. Finding order statistics

Problem Select
Input: a list L and rank k;
Output: the k^{th} smallest element in L;

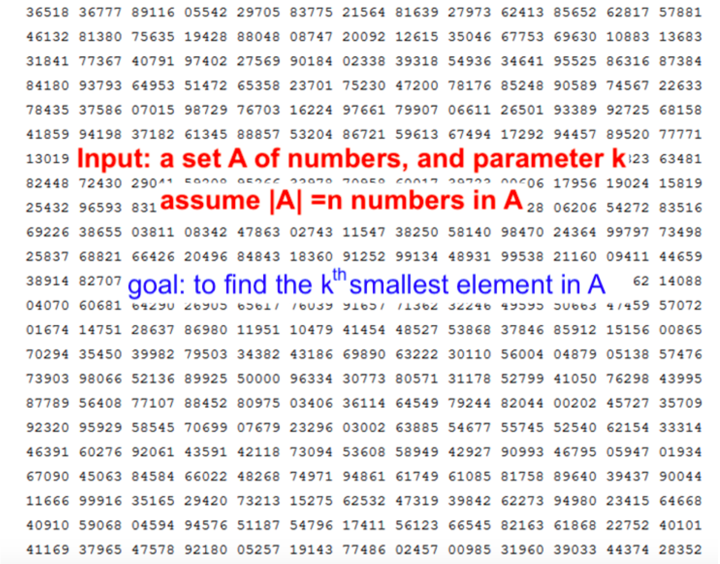
- with sorting, Select problem can be done in time $O(n \log_2 n)$;
- can we do better – in $O(n)$ time ?
- Select problem for $k = 1, n$ is easy: $O(n)$;
- how about finding the 2nd largest element?

Two select algorithms

- deterministic algorithm: worst case in linear time;
- randomized algorithm: averaged case in linear time;

4. Finding order statistics

Idea of linear time deterministic `select(A,n,k)` algorithm:



4. Finding order statistics

Idea of linear time deterministic `select(A,n,k)` algorithm:

36518	36777	89116	05542	29705	83775	21564	81639	27973	62413	85652	62817	57881
46132	81380	75635	19428	88048	08747	20092	12615	35046	67753	69630	10883	13683
31841	77367	40791	97402	27569	90184	02338	39318	54936	34641	95525	86316	87384
84180	93793	64953	51472	65358	23701	75230	47200	78176	85248	90589	74567	22633
78435	37586	07015	98729	76703	16224	97661	79907	06611	26501	93389	92725	68158
41859	94198	37182	61345	88857	53204	86721	59613	67494	17292	94457	89520	77771
13019	07274	51068	93129	40386	51731	44254	66685	72835	01270	42523	45323	63481
82448	72430	29041	59208	95266	33978	70958	60017	39723	00606	17956	19024	15819
25432	96593	83112	96997	55340	80312	78839	09815	16887	22228	06206	54272	83516
69226	38655	03811	08342	47863	02743	11547	38250	58140	98470	24364	99797	73498
25837	68821	66426	20496	84843	18360	91252	99134	48931	99538	21160	09411	44659
38914	82707	24769	72026	56813	49336	71767	04474	32909	74162	50404	68562	14088
04070												57072
01674												00865
70294	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	57476
73903	98066	52136	89925	50000	96334	30773	80571	31178	52799	41050	76298	43995
87789	56408	77107	88452	80975	03406	36114	64549	79244	82044	00202	45727	35709
92320	95929	58545	70699	07679	23296	03002	63885	54677	55745	52540	62154	33314
46391	60276	92061	43591	42118	73094	53608	58949	42927	90993	46795	05947	01934
67090	45063	84584	66022	48268	74971	94861	61749	61085	81758	89640	39437	90044
11666	99916	35165	29420	73213	15275	62532	47319	39842	62273	94980	23415	64668
40910	59068	04594	94576	51187	54796	17411	56123	66545	82163	61868	22752	40101
41169	37965	47578	92180	05257	19143	77486	02457	00985	31960	39033	44374	28352

Numbers in A are grouped into n/5 groups, with 5 numbers in each group

4. Finding order statistics

Idea of linear time deterministic `select(A,n,k)` algorithm:

36518	36777	89116	05542	29705	83775	21564	81639	27973	62413	85652	62817	57881
46132	81380	75635	19428	88048	08747	20092	12615	35046	67753	69630	10883	13683
31841	77367	40791	97402	27569	90184	02338	39318	54936	34641	95525	86316	87384
84180	93793	64953	51472	55358	23701	75230	47200	78176	85248	90589	74567	22633
78435	37586	07015	98729	76703	16224	97661	79907	06611	26501	93389	92725	68158
41859	94198	37182	61345	88857	53204	86721	59613	67494	17292	94457	89520	77771
13019	07274	51068	93129	40386	51731	44254	66685	72835	01270	42523	45323	63481
82448	72430	29041	59208	95266	33978	70958	60017	39723	00606	17956	19024	15819
25432	96593	83112	96997	55340	80312	78839	09815	16887	22228	06206	54272	83516
69226	38655	03811	08342	47863	02743	11547	38250	58140	98470	24364	99797	73498
25837	68821	66426	20496	84843	18360	91252	99134	48931	99538	21160	09411	44659
38914	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
04070	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
01674	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
70294	35450	39982	79503	34382	43186	69890	63222	30110	56004	04879	05138	57476
73903	98066	52136	89925	50000	96334	30773	80571	31178	52799	41050	76298	43995
87789	56408	77107	88452	80975	03406	36114	64549	79244	82044	00202	45727	35709
92320	95929	58545	70699	07679	23296	03002	63885	54677	55745	52540	62154	33314
46391	60276	92061	43591	42118	73094	53608	58949	42927	90993	46795	05947	01934
67090	45063	84584	66022	48268	74971	94861	61749	61085	81758	89640	39437	90044
11666	99916	35165	29420	73213	15275	62532	47319	39842	62273	94980	23415	64668
40910	59068	04594	94576	51187	54796	17411	56123	66545	82163	61868	22752	40101
41169	37965	47578	92180	05257	19143	77486	02457	00985	31960	39033	44374	28352

The 3rd largest number in every group is chosen;
all such numbers are placed in new set M

4. Finding order statistics

Idea of linear time deterministic `select(A,n,k)` algorithm:

$M =$

51731	44254	66685	72835	01270
33978	70958	60017	39723	00606
80312	78839	09815	16887	22228
02743	11547	38250	58140	98470
18360	91252	99134	48931	99538
49336	71767	04474	32909	74162
76039	91657	71362	32246	49595
10479	41454	48527	53868	37846

$|M| = n/5$ elements, e.g., 40

4. Finding order statistics

Idea of linear time deterministic `select(A,n,k)` algorithm:

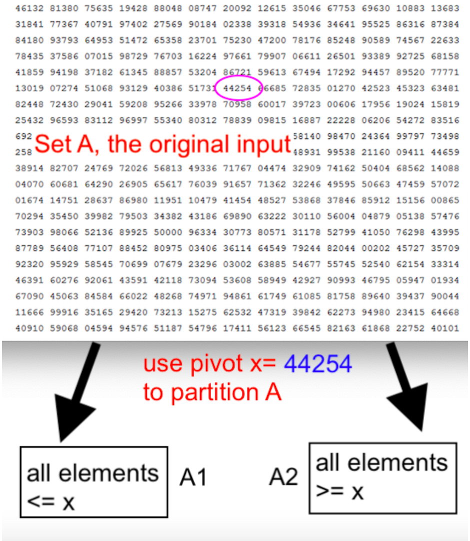
$M =$

51731	44254	66685	72835	01270
33978	70958	60017	39723	00606
80312	78839	09815	16887	22228
02743	11547	38250	58140	98470
18360	91252	99134	48931	99538
49336	71767	04474	32909	74162
76039	91657	71362	32246	49595
10479	41454	48527	53868	37846

Find $(\frac{n}{10})^{\text{th}}$ smallest element, e.g., the 20^{th} if $|M| = n/5 = 40$
let this element be 44254, name it x, the *pivot*

4. Finding order statistics

Idea of linear time deterministic `select(A,n,k)` algorithm:



4. Finding order statistics

Idea of linear time deterministic `select(A,n,k)` algorithm:

Summary of the steps described so far:

- **input**: list A of n elements, and parameter k;
  (goal: to find k^{th} smallest element from A)
- group elements in A into groups of 5, resulting in $n/5$ groups;
- for each group, pick the third largest element;
  put such elements from all groups in M;
- let $x =$ `select(M, n/5, n/10)`, a recursive call to `select`;
  x is the $(\frac{n}{10})^{\text{th}}$ smallest element in M, or median in M;
- use x as pivot to partition A into A_1 and A_2, such that
  $\forall y \in A_1, y \leq x$, and $\forall z \in A_2, x < z$

4. Finding order statistics

Idea of linear time deterministic `select(A,n,k)` algorithm:

Continue the idea:

- let $r = rank(x)$ in A, i.e., x is the r^{th} smallest element in A;
- if $r = k$ (remember k?), then the algorithm **returns** x, done!
- otherwise,

  if $k < r$, **return** `select(A1, r-1, k)`;

  else, **return** `select(A2, n-r, k-r)`;

4. Finding order statistics

Algorithm $\textsc{Select}(A, n, k)$; $\longleftarrow$ finding the k^{th} smallest element in A

1. **if** $n < 100$, simply sort A and **return** $A[k]$;
2. Find a pivot
3. group elements in A into groups of 5 elements;
4. place 3^{rd} largest elements from all groups in list M;
5. $x = \textsc{Select}(M, \frac{n}{5}, \frac{n}{10})$;
6. let $r = rank(x)$ in A;
7. **if** $k = r$, return $A[r]$;
8. partition A into $A_1 = \{y : y \leq x\}$, $A_2 = \{z : x < z\}$;
9. **if** $k < r$, **return** $\textsc{Select}(A_1, r - 1, k)$;
10. **else return** $\textsc{Select}(A_2, n - r, k - r)$;

4. Finding order statistics

Assume $T(n)$ to be the time function for $\textsc{Select}(A, n, k)$.

Algorithm $\textsc{Select}(A, n, k)$; $\longleftarrow$ time function $T(n)$ on n elements of A

1. **if** $n < 100$, simply sort A and **return** $A[k]$; $\longleftarrow c_1$
2. Find a pivot
3. group elements in A into groups of 5 elements; $\longleftarrow \leq c_3 n$
4. place 3^{rd} largest elements from all groups in list M; $\longleftarrow \leq c_4 n$
5. $x = \textsc{Select}(M, \frac{n}{5}, \frac{n}{10})$; $\longleftarrow T(\frac{n}{5})$
6. let $r = rank(x)$ in A; $\longleftarrow \leq c_6 n$
7. **if** $k = r$, return $A[r]$; $\longleftarrow c_7$
8. partition A into $A_1 = \{y : y \leq x\}$, $A_2 = \{z : x < z\}$; $\longleftarrow \leq c_8 n$
9. **if** $k < r$, **return** $\textsc{Select}(A_1, r-1, k)$; $\longleftarrow T(|A_1|)$
10. **else return** $\textsc{Select}(A_2, n-r, k-r)$; $\longleftarrow T(|A_2|)$

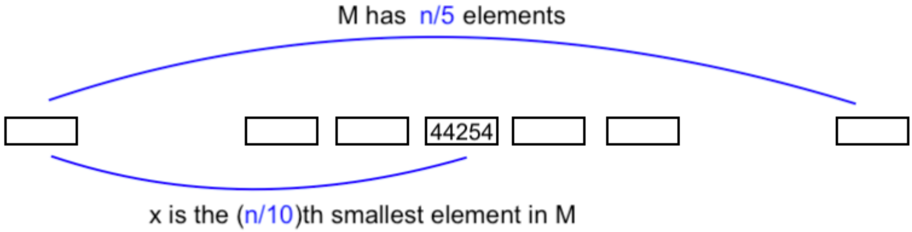
how small can they be?

$$T(n) \leq \begin{cases} c_1 & n < 140 \\ T(\frac{n}{5}) + \max\{T(|A_1|), T(|A_2|)\} + an + b & n \geq 140 \end{cases}$$

4. Finding order statistics

44254

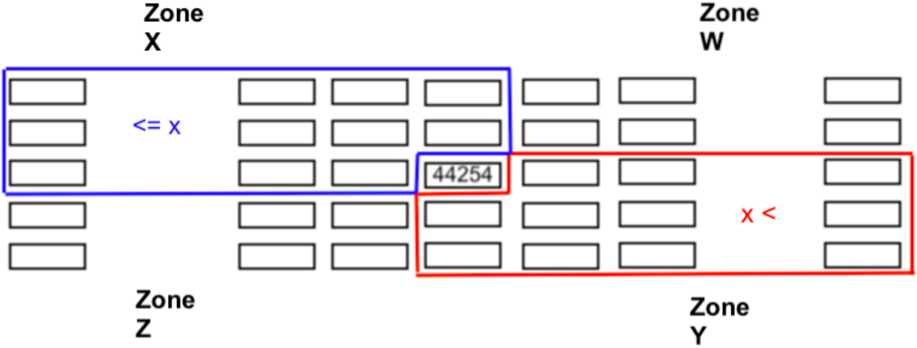
4. Finding order statistics



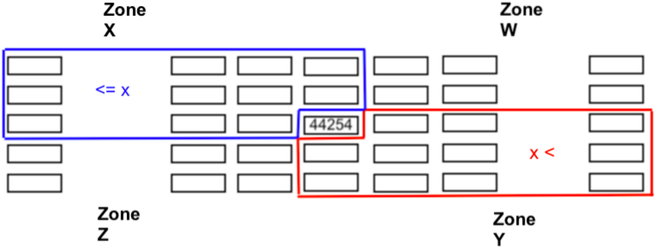
4. Finding order statistics

4. Finding order statistics

4. Finding order statistics



4. Finding order statistics



- $|X| = \frac{n}{10} \times 3 - 1; \Longrightarrow |A_1| \geq \frac{3n}{10} - 1; \Longrightarrow |A_2| = n - |A_1| - 1 \leq \frac{7n}{10};$
- $|Y| = \frac{n}{10} \times 3 - 1; \Longrightarrow |A_2| \geq \frac{3n}{10} - 1; \Longrightarrow |A_1| = n - |A_2| - 1 \leq \frac{7n}{10};$
- $\max\{|A_1|, |A_2|\} \leq \frac{7n}{10};$

$$T(n) \leq \begin{cases} c_1 & n < 100 \\ T(\frac{2n}{10}) + T(\frac{7n}{10}) + an + b & n \geq 100 \end{cases}$$

4. Finding order statistics

Deterministic select has time complexity:

$$T(n) \leq \begin{cases} c_1 & n < 100 \\ T(\frac{2n}{10}) + T(\frac{7n}{10}) + an + b & n \geq 100 \end{cases}$$

Use strong math induction to prove that $T(n) = O(n)$, or equivalently, there is $c > 0$, $T(n) < cn$ for all $n \geq 1$;

Taken-home exercises II(B)

1. What is the main technical difference in deterministic quick sort and randomized quick sort?
2. What is the worst case time complexity of deterministic quick sort? How about randomized quick sort?
3. What is the average case time complexity of randomized quick sort? Can the deterministic quick sort have average case time complexity? if so, under what condition?
4. Though the `partition` subroutine has the worst case time complexity $O(n)$ on input list of n elements, it may execute different numbers of basic operations on different lists. What are the worst cases of input lists that may incur the maximum number of basic operations for `partition` to execute?
5. Assume the input list to quick sort contains duplicated elements. Design a subroutine `partition-3(L, low, high)` that chooses the last element `L[high]` as pivot e to partition the list into 3 segments, which contain elements $< e$, $= e$, and $> e$, respectively. The subroutine returns two indexes that mark the boundaries between the 3 segments. It is required that the subroutine uses only one single **for** loop, in the spirit of function `partition` for quick sort.

Taken-home exercises II(B)

5. Consider to sort lists of (length n) that contain at most k distinct elements, for $k \ll n$ (i.e., k is much smaller than n). For example, $n = 20$, $k = 5$, such a list may look like: `(4,8,3,4,4,3,2,8,2,4,3,9,8,8,4,9,4,4,2,9)`, which contains three 2's, three 3's, seven 4's, four 8's and three 9's.
   Assume that a quick sort algorithm removes duplicated elements as the pivot before the next rounds of recursive calls (e.g., via the above `partition-3` subroutine), which of the followings is the correct worst case time complexity of the modified quick sort on such input lists?Justify your answer.

   (1) $O(nk)$, (2) $O(n \log_2 k)$, (3) $O(k \log_2 n)$, (4) $O(k \log_2 k)$

6. Consider algorithm `select` that finds the k^{th} smallest element from the unsorted input list. If, groups of 7 elements, instead of 5, are used by `select`, derive a recurrence relation for its worst case time complexity $T(n)$.
7. Again for algorithm `select`, if groups of 3 elements, instead of 5, are used by `select`, derive a recurrence relation for its worst case time complexity $T(n)$.

Taken-home exercises II(B)

8. What algorithms have the following recurrences for running time?
   - $T(n) = T(n-1) + b$;
   - $T(n) = T(n-1) + bn$;
   - $T(n) = T(\frac{n}{2}) + b$;
   - $T(n) = dT(\frac{n}{2}) + bn$; $d = 2, 3, 4$;
   - $T(n) = dT(\frac{n}{2}) + bn^2$; $d = 7, 8$;
   - $T(n) = T(\alpha n) + T(\beta n) + bn$, for $\alpha + \beta = 1$
   - what about: $T(n) = T(\alpha n) + T(\beta n) + T(\gamma n) + bn$, for $\alpha + \beta + \gamma = 1$
   - what about $d = 1$, i.e., $T(n) = T(\frac{n}{2}) + bn$;
   - more general $T(n) = T(\alpha n) + T(\beta n) + bn$, for $\alpha + \beta < 1$