CS 341 – Algorithms

Lecture 3 – Divide and Conquer

19 May 2021

Today’s Plan

1. Counting Inversion Pairs

2. Maximum Sum Subarray

3. Computing the Median

Counting Inversion Pairs

Input: n distinct numbers $a_1, a_2, \dots, a_n$

Output: the number of "inversion" pairs with $i < j$ but $a_i > a_j$

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5$

e.g. $3, 7, 5, 1, 4$

inversion pairs $(3,1), (7,1), (7,5), (7,4), (5,1), (5,4)$

6 inversion pairs.

naive: $\Theta(n^2)$

Divide and Conquer

Again assume that n is a power of two.

3	7	2	5	9	1	6	4

suppose counted # of inversion pairs in each half

3	7	2	5

3 pairs

9	1	6	4

4 pairs

it remains to count the number of "crossing" inversion pairs

i.e i left, j right

obs: easier, because relative positions are fixed

3	7	2	5

0	4	1	2
9	1	6	4

$\Rightarrow$ crossing inversion pairs $= 7$

\# of crossing inversion pairs involving 1 = # of elements on the left greater than 1

Counting "Crossing" Inversion Pairs

How to count the number of crossing inversion pairs efficiently?

idea 1: sort the numbers

2	3	5	7

4	2	1	0
1	4	6	9

use binary search $O(\log n)$ total time: $O(n \log n)$

$T(n) = 2T(\frac{n}{2}) + O(n \log n)$ ← sorting + binary search

$\Rightarrow T(n) = O(n \log^2 n)$

idea 2: merge sort

2	3	5	7

1	4	6	9

$O(n)$

+4			+2		+1		+0	
1	2	3	4	5	6	7	9	

Recursive Algorithms

count $(A[1,n])$ $T(n)$

base case : return ;

count $(A[1,\frac{n}{2}])$ $T(\frac{n}{2})$

count $(A[\frac{n}{2}+1,n])$ $T(\frac{n}{2})$

sort $(A[1,\frac{n}{2}])$ $n\log n$

sort $(A[\frac{n}{2}+1,n])$ $n\log n$

merge-count n

$$T(n) \le 2T(\tfrac{n}{2}) + O(n\log n)$$

$$\Rightarrow T(n) = O(n\log^2 n)$$

count-sort $(A[1,n])$

base case : $n=1$, return 0.

$S_1 =$ count-sort $(A[1,\frac{n}{2}])$ $T(\frac{n}{2})$

$S_2 =$ count-sort $(A[\frac{n}{2}+1,n])$ $T(\frac{n}{2})$

$S_3 =$ merge-count $O(n)$

↑

sort

← exercise

return $S_1+S_2+S_3$

$$T(n) \le 2T(\tfrac{n}{2}) + O(n)$$

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

just a slight modification of merge sort!

Today’s Plan

1. Counting Inversion Pairs
2. Maximum Sum Subarray
3. Computing the Median

Maximum Sum Subarray

Input: n numbers $a_1, a_2, \ldots, a_n$

Output: i, j that maximizes $\sum_{k=i}^{j} a\_k$

$i = j \quad j \quad j \quad j \qquad j$

e.g. $3, -4, \underbrace{5, 6, 1, -10, 2, 7, 8, -13, 7, 7}$

naive: try i, j, compute $\sum_{k=i}^{j} a_k$ Time: $\Theta(n^3)$

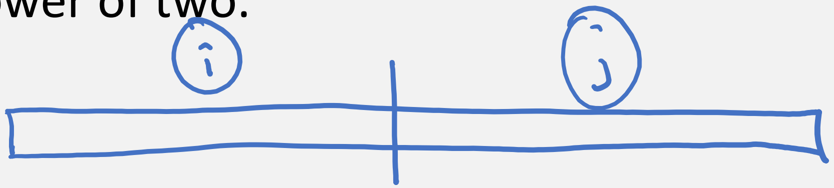
better: try i n iterations

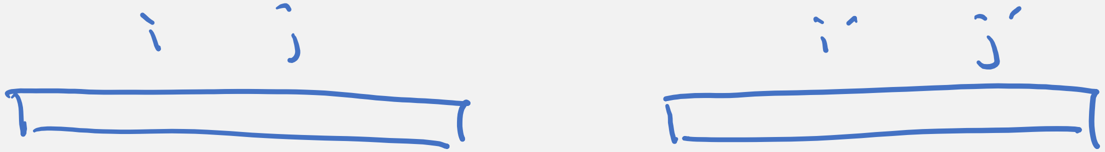
compute $\sum_{k=i}^{j} a_k$ for all j $\leftarrow$ $O(n)$ time

Time: $\Theta(n^2)$

Divide and Conquer

Again assume that n is a power of two.

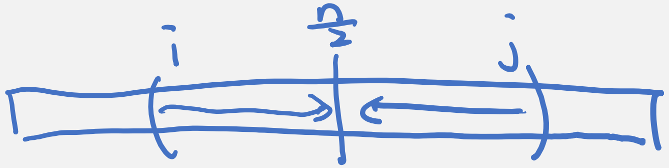




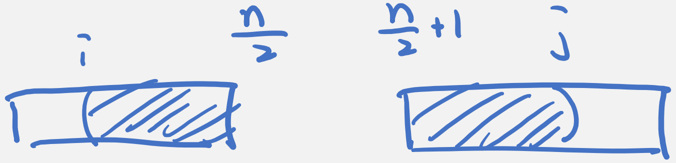
max sum subarray on left

solved on right

remains to find the maximum "crossing" sum



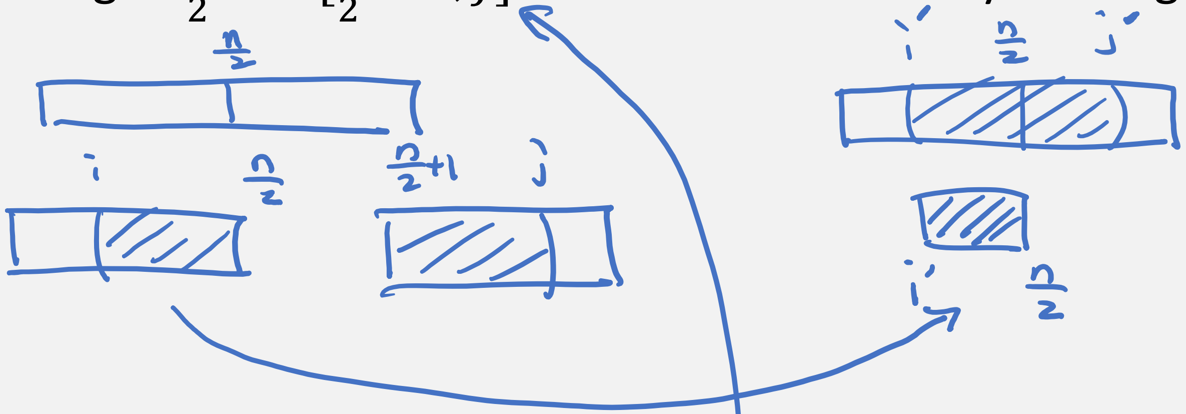
obs: easier because the midpoint is fixed



Maximum Crossing Sum

Claim. For $i \le \frac{n}{2} < j$, $[i, j]$ is a maximum sum subarray crossing the mid-point if and only if

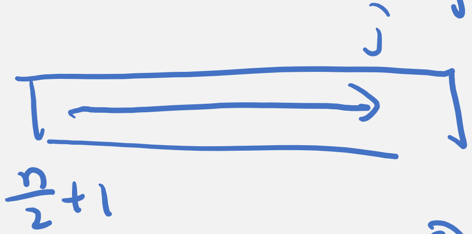
$[i, \frac{n}{2}]$ is a max sum subarray ending at $\frac{n}{2}$ and $[\frac{n}{2}+1, j]$ is a max sum subarray starting at $\frac{n}{2}+1$.



partial sum starting at $\frac{n}{2}$

$O(n)$

partial sum starting at $\frac{n}{2}+1$



$O(n)$

Time Complexity and Questions

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \leftarrow \text{cross sum}$$

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

Challenge: Design a $O(n)$ time algorithm for the maximum sum subarray problem.

greedy , dynamic programming

Today’s Plan

1. Counting Inversion Pairs
2. Maximum Sum Subarray
3. Computing the Median

Finding the Median

Input: n distinct numbers $a_1, a_2, \ldots, a_n$

Output: the median of these numbers

sort : $O(n \log n)$ time

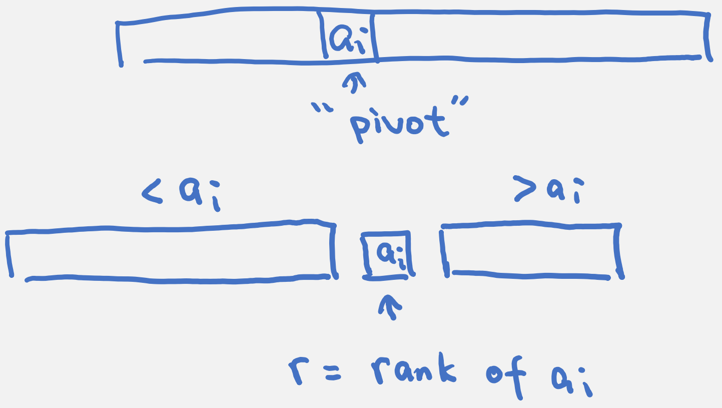
Input: n distinct numbers $a_1, a_2, \ldots, a_n$ and an integer $k \leq n$

Output: the k-th smallest number in $a_1, a_2, \ldots, a_n$

easier for recursion

Divide and Conquer

The idea is similar to that in quicksort (which is a divide and conquer algorithm).



unsorted

$O(n)$ time to divide into 3 groups

if $r=k$ - done.

if $r>k$, then find the k-th smallest element on the left

Time : $T(r-1)$

if $r<k$, then find the $(k-r)$-th smallest element on the right

Time : $T(n-r)$

Finding a Good Pivot

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

e.g. $r = n$, then $T(n) \leq T(n-1) + O(n) \Rightarrow T(n) = O(n^2)$.

Suppose $r = \frac{n}{2}$, then $T(n) \leq T(\frac{n}{2}) + \underbrace{P(n)}_{\text{find a good pivot}} + O(n) \Rightarrow T(n) = O(n)$ if $\underbrace{P(n) = O(n)}_{\text{time to median}}$

Suppose $\underbrace{\frac{n}{10} \leq r \leq \frac{9n}{10}}_{\text{good pivot}}$, then $T(n) \leq T(\frac{9n}{10}) + P(n) + O(n)$

$\Rightarrow T(n) = O(n)$ if $P(n) = O(n)$

We have made some progress in the median problem, by reducing the problem of finding the middle number to the easier problem of finding a number not too far from the middle.

Median of Medians

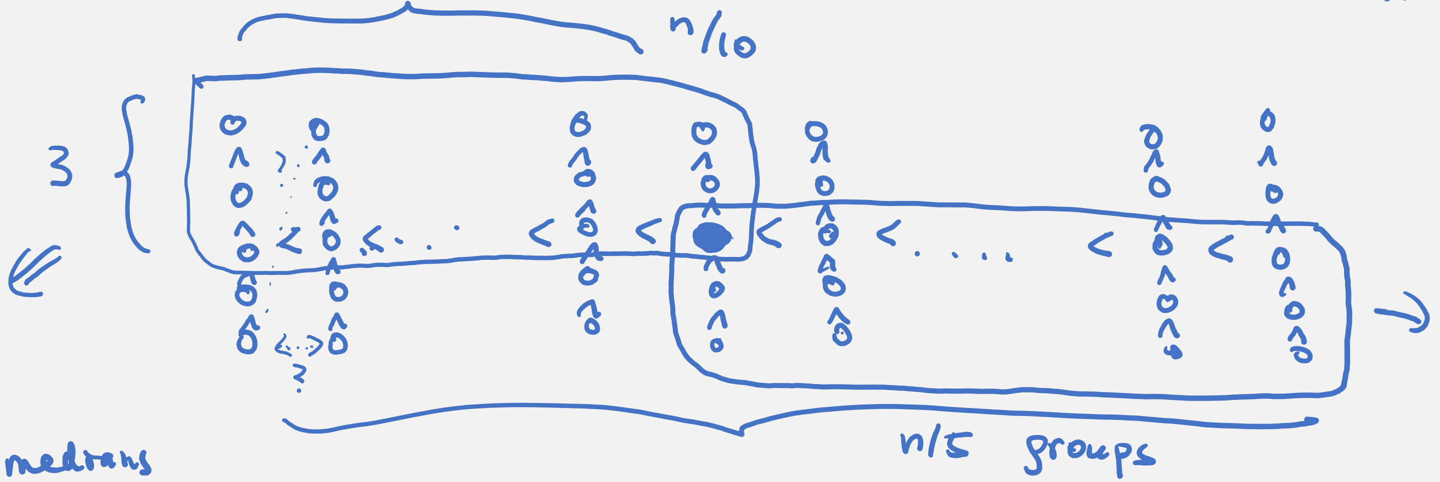
① divide the n elements into $\frac{n}{5}$ groups. each group has 5 numbers   Time: $O(n)$

② compute the median of each group. call them $b_1, b_2, \dots, b_{\frac{n}{5}}$ Time: $O(n)$

③ return the median of these medians. $b_1, b_2, \dots, b_{\frac{n}{5}}$ Time: $T(\frac{n}{5})$

Lemma   Let r be the rank of median of medians. Then $\frac{3n}{10} \le r \le \frac{7n}{10}$.

Proof



for analysis

$\geq \frac{3n}{10}$ elements smaller than the median of medians

$\frac{3n}{10}$ elements bigger than the output

□

Time Complexity

$P(n) \le T(\frac{n}{5}) + c_1 n \leftarrow$ median of each group

$T(n) \le \underbrace{P(n)}_{\text{good pivot}} + \underbrace{T(\frac{7n}{10})}_{\text{because of splitting using good pivot}} + c_2 n \leftarrow$ splitting.

$\le T(\frac{n}{5}) + T(\frac{7n}{10}) + c_1 n + c_2 n$

L02 $\Rightarrow$ $T(n) = O(n)$. $\square$

uneven subproblems

Exercise: What if we divide into groups of 3, 7, 9,…,$\sqrt{n}$? $\leftarrow$ Alex tutorial 2

Exercise: Unfold the recursion to understand the algorithm better.