CS3000: Algorithms & Data
Paul Hand

Lecture 7:

- Divide and Conquer Example – Similar to HW
- Binary Search
- Another Example

Jan 30, 2019

Practice Problem:
Maximum Sum Subarray Problem

Maximum Sum Subarray Problem

- Input: Array A[1:n] of integers
- Problem: Find a subarray A[i:j] with the largest possible sum
- Example: A = [3, -4, $\underbrace{5, -2, -2, 6, -3, 5}_{\text{Maximum sum = 9}}$, -3, 2]

Step 1: Find a "naive" method

Q: If we dont care about efficiency how can we solve?
How bad is that approach?
(what is the time complexity of it)

Approach: "try everything"
- Iterate over all subarrays
Choose max

For i = 1 ... n — one factor of n
For j = i+1 ... n — factor of n
Evaluate sum A[i:j] — at most n terms in sum
return max

complexity is $\Theta(n^3)$

Feels very slow.

Try divide & conquer

- Input: Array A[1:n] of integers
- Problem: Find a subarray A[i:j] with the largest possible sum
- Task: Devise a divide and conquer algorithm to solve this problem. Consider an algorithm that divides A into two halves.
- Task: Devise a divide and conquer algorithm to solve this problem. Consider an algorithm that divides A into two halves.

What do I need to get divide + conquer to work?

Base case ✓

If I've solved two smaller ~~[illegible]~~ versions how do I "merge" them together efficiently?

A = [n/2][n/2]

know best subarray among these

know best subarray here too

Can I acquire the best subarray over the whole problem efficiently? O(n)?

wonder: what hasn't been considered so far.

Subarrays that live in both halves

O(n) choices for start

O(n) choices for finish

Could evaluate all O(n) subarrays living in right half (containing) in O(n) time

[x][x]

Subarray that spans both sides has left part & right part

Separately find best subarray on left & best subarray on right
(includes middle elts)

[[x]
[[xx]
[[xxx]
⋮
[[xxxxxx]

O(n) diff choices
can be evaluated in O(n) time.

~~At~~ (A) Best subarray lives only on Left

(B) Best subarray lives only on Right

(C) Best subarray lives on both

can be found by best subarray living on R that contains leftmost entry of R

+

best subarray living on L that contain rightmost entry of L

Choose best among (A) (B) (C).

Divide-and-Conquer: Binary Search

Binary Search

anything left is < 28. Dont look at it

Is 28 in this list?

~~2~~	~~3~~	~~8~~	11	15	17	28	42	A

Naive alg: linear search. check A[i] for i=1...n. O(n) time. Bad

did not exploit structure

~~15~~	17	28	42

28	42

got 28. in list.

Binary Search

```
Search(A,t):
  // A[1:n] sorted in ascending order
  Return BS(A,1,n,t)
```

left end of "active" region

```
BS(A,ℓ,r,t):
```

right end of "active" region

```
  If(ℓ > r): return FALSE
```

width

$m \leftarrow \ell + \left\lfloor \frac{r-\ell}{2} \right\rfloor$ midpoint of list & round down

nothing to right of m matters

```
  If(A[m] = t): Return m
  ElseIf(A[m] > t): Return BS(A,ℓ,m-1,t)
  Else: Return BS(A,m+1,r,t)
```

modify right endpoint

Activity

- What is the running time of binary search?
  - What is the recurrence?
  - What is the solution to the recurrence?

```
Search(A,t):
  // A[1:n] sorted in ascending order
  Return BS(A,1,n,t)

BS(A,ℓ,r,t):
  If(ℓ > r): return FALSE
```

$$m \leftarrow \ell + \left\lfloor \frac{r-\ell}{2} \right\rfloor$$

```
  If(A[m] = t): Return m
  ElseIf(A[m] > t): Return BS(A,ℓ,m-1,t)
  Else: Return BS(A,m+1,r,t)
```

effective/active part of A is A[r:ℓ]

$\boxed{\ell - r + 1 = n}$

$O(1)$

$T(n) = T(\frac{n}{2}) + Cn^0$ $\quad T(\frac{n}{2})$

no 2 here b/c didn't recurse on both halves.

Master thm: $a = 1$, $b = 2$, $d = 0$

$\frac{a}{b^d} = 1$

By thm?

$T(n) = \Theta(n^d \log n)$

$= \Theta(\log n)$

Proof of Correctness for Binary Search

```
Search(A,t):
  // A[1:n] sorted in ascending order
  Return BS(A,1,n,t)

BS(A,ℓ,r,t):
  If(ℓ > r): return FALSE
```

$$m \leftarrow \ell + \left\lfloor \frac{r-\ell}{2} \right\rfloor$$

```
  If(A[m] = t): Return m
  ElseIf(A[m] > t): Return BS(A,ℓ,m-1,t)
  Else: Return BS(A,m+1,r,t)
```

Proof of Correctness of Binary Search

Clm: $\forall n \in \mathbb{N}$ $\forall\ \ell, r$ s.t. $r-\ell \le n$, $\forall A$, $\forall t$

$$BS(A,\ell,r,t) = \begin{cases} i \text{ s.t. } A[i]=t \\ \perp \text{ if } t \notin A \end{cases}$$

False

H(n)

Inductive Hyp

H(0) ...

Base Case: H(1) the algorithm is correct

Inductive Step: Assume H(n) is true

Suppose that we get $BS(A, \ell, r, t)$ and $r-\ell \le n+1$

$m \leftarrow \ell + \lfloor \frac{r-\ell}{2} \rfloor$

(case 1) If $A[m]=t$ ✓

(case 2) $A[m] > t \Rightarrow t$ is not in $A[m:n]$ b/c list was sorted.

By I.H., $BS(A,\ell,m-1,t) = i$ if $t \in A[\ell:m-1]$ and $\perp$ otherwise

If returns false, $t \notin A[\ell:m-1]$. And $t \notin A[m:n]$ $\Rightarrow t \notin A$.

(case 3) $A[m] < t$.

Same as case 2.

Binary Search Wrapup

- Search a sorted array in time $O(\log n)$!!!
- Divide-and-conquer approach
  - Find the middle of the list, recursively search half the list
  - **Key Fact:** eliminate half the list each time
- Prove correctness via induction
- Analyze running time via recurrence
  - $T(n) = T(n/2) + C$

If we want to search many times, worth it to sort in advance

Q: If I want to check if $t \in$ list A, is it worth it to sort A and do binary search??

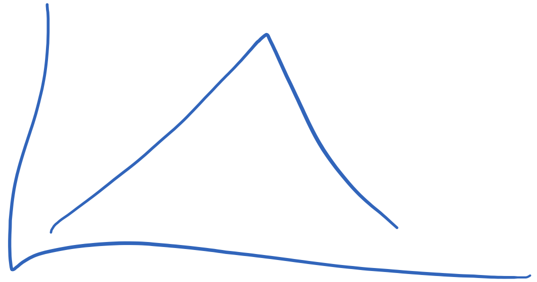
Sort: $n \log n$ search: $\log n$ $n \log n + \log n = \Theta(n \log n)$

Practice Problem:
Finding maximum of unimodal list

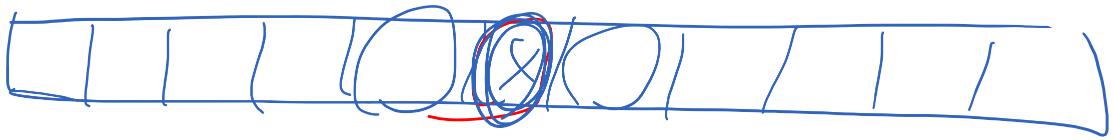
Max of Unimodal List

no repeats

- Input: Array A[1:n] of integers. A[1:i] is strictly increasing. And A[i+1:n] is strictly decreasing.
- Problem: Find largest element in O(log(n)) time
- Examples:
  A = [1,4,5,3,0]
  A = [5,2,1,0,-2]
  A = [2,4,7,9]
  all unimodal



Inspired by binary search
how could you solve it by
divide + conquer?



Naive alg? O(n) time.
Try all possibilities

If x is max among these 3, return x.
If left is max, throw away what's right
If right is max, — — — left

Max of Unimodal List

- Input: Array A[1:n] of integers. A[1:i] is increasing. And A[i+1:n] is decreasing.
- Problem: Find largest element in O(log(n)) time
- Examples:
  A = [1,4,5,3,0]
  A = [5,2,1,0,-2]
  A = [2,4,7,9]