CS 513 Data Structure and Algorithm

Martin Farach-Colton

Core 313 Friday 10:30-11:30

TA: Vishvajeet N [vishva.jeet@rutgers.edu](mailto:vishva.jeet@rutgers.edu)

# September 8, 2017

Maximum Subvector Sum

-10 -20 10 5 -3 -2 -1 6 4 -3 -16 32 12 -1

Output (12,13)

-10 -20 10 5 -3 -2 -1 6 4 -3 -16 -32 12 -1

Output (3,9)

Current Max

0 0 10 15 12 10 9 15 19 16 0 0 12 11

Def we say f(n) is O(g(n)) if there exists C, n0 that for any n>n0, f(n)<=cg(n)

Eg Def f(n) is O(g(n)) if lim f(n)/g(n) <=c, for some constant c

F(n)=3123n^3+17.5n^2-321n+1077.98

Find g(n) that f(n)=O(g(n))

g(n^4) can be the answer where c=0

Def f(n) is Ω(g(n)) if g(n) is O(f(n)) Ωgreater than, big-O smaller than

The big O notation has nothing to do with algorithms. Only with funcitons.

Def f(n) is Θ(g(n)) if f(n) is O(g(n)) and f(n) is Ω(g(n)) or lim f(n)/g(n)=c>0

L’Hosital’s Rule

Lim f(n)/g(n)=lim f’(n)/g’(n) (also true when n->0)

Product rule when taking derivatives

Multiplying Integers

X\*Y n bits

Long multiplication is O(n^2)

X-A\*2^(n/2)+B

B is low order bits (latter half of bits)

A is high order bits (first half of bits)

Y=C\*2^(n/2)+B

X\*Y=AC\*2^n+(AD+BC)\*2^(n/2)+BD plus a base case to form a recursive

T(n)=4T(n/2)+O(n)

Shifting, adding

Master’s theorem

The recurrence T(1)=c

T(n)=aT(n/b)+df(n)

For multiplicative f is

Θ(n^()) if a>f(b)

Θ(f(n)) If a<f(b)

Θ(n^() logn) If a=f(b)

Def f is multiplicative if f(ab)=f(a)f(b) Ex. F(n)=2n is not, f(ab)=2ab!=f(a)f(b) . f(n)=n is

In master’s theorem f(n) is multiplicative thanks to constant d

The recursive equation running time is O(n^2)

(A+B)(C+D)-AC-BD=AD+BC identity (no longer need to consider AD BC multiply)

The equation was simplified to (3 multiply instead of 4)

T(n)=3T(n/2)+O(n)

a=3 f(b)=2 O(n^(log 3))

Sorting

Bubble sort: running time n(n-1)/2 O(n^2)

Below n^2 and above n so Ω(n)

Worst case W(n)=Θ(n^2) a tight bound

(1). Worst-case running time

Let Dn={all inputs of size n}

For I in Dn, let T(I) be there running time on input I

Then worst case running time w(n)=max {T(I)} I in Dn

(2). Average-case

Let p(I)= probability of input I being given

Then A(n)= weighted average

Dn= set of permutations

|Dn|=n! (size of Dn)

p(I)=1/n!

An inversion in a permutation is a pair that is out of order

5,3,1,2,4

Inversion: 5,3

5,1

5,2

5,4

3,1

3,2

Every swap in bubble sort removes exactly 1 inversion

Average running time of bubble sort >= average # of inversion in a uniformly selected permutation

Comparison + swap swap only

4,2,1,3,5

# of inversion in (π) + # of inversion in () = (R means reverse)

Because every pair is an inversion in π or (not both)

Average # inversion in a permutation is /2

A(n)=Ω(n^2) at least lower bound, because is n^2

But w(n)=Θ(n^2) so A(n)=Θ(n^2)

Sorting algorithms

1. Comparison based vs not

Comparison based: Merge quick bubble insertion

Non-comparison:

Micheal Fragmon rumble memory fusion trees

1. In place =O(1) extra memory (besides the input array)

Bubble in place

Quick sort recursive cannot be in place because you are using the stack

How deep in quick sort can your recursive stack be ? O(log n)

1. Stable = you do not change the order of repeated elements ()

Merge sort (stable if when you are merging always choose the left array in a tie)

Quick sort (not stable partition)

A=[a1,a2,a3…..an]

(a1,1)(a2,2) if the first field is the same, check the second field, making it stable

1. Internal RAM vs External Disk

(Sorting data may be too big for RAM)

Next lecture:

Proof of lower bound of comparison-based sorting is nlogn

# September 15, 2017

Disk Access Model (DAM)

CPU 🡪 Memory M array

B

Reads Size B I/O each time to memory

I/Os are 5 to 6 orders of magnitude slower than instructions.

Since I/Os are expensive

Compression is the general thing with more CPU and less I/Os

Analyze Merge sort in DAM:

I/Os to merge two arrays of size N : O(N/B)

N divided by B (every I/O fetch B)

T(N)=2T(N/2)+N/B

T(M)=M/B

B is not a constant. So can’t apply master theorem. For some case, N, B, M can go infinity. None of them can be taken as constant.

Solve in Tree: N->N/2->N/4->……->N/B == M

Each level is N/B

Height is logN- logM = log(N/M)

Total is O(N/B\*log(N/M))

How to optimize this algorithm?

Do K-way Merge sort.

T(N)=KT(N/K)+N/B is the same for some K, need to be able to fit K+1 blocks in memory

Tree: N🡪 N/K🡪N/K^2🡪N/K^3🡪…..M

O()

K<=M/B fits K+1 blocks in memory

O()

It’s so ugly but we cannot do better.

This is the correct algorithm for sorting in external memory.

2-way merge sort is optimum in external sorting

Nothing has one level of memory , call cache

Higher way merge sort

Word-RAM\_ operations on words take O(1) time and words have w bits

What do we know about the size of w?

From theoretical point of view, you think your word size should not increase with your data size.

A[i]=j should be constant time O(1). Then how many bits in word?

How big can A be? Say the length of A has n items

How do I find the ith element in an array?

This is actually base A+i and multiply the size of the item.

How many bits to distinguish between N items? Log N

w>= log n

In constant time when w is at least log n bits.

Non-comparison based

Bucket sort Sort array A drawn from a universe u=[0,u-1}=[u-1]

Not in place, but stable

B[A[i]]

u aray tree ???

Runtime O(n+u) that does not mean sort in linear time. If you know u is small, you could use this algorithm instead of merge sort.

Radix sort

Stable sort by least significant digit

If we do in bucket sort. If there is K digit, each digit has [u-1]. Then it’s n by K array. O(K(n+u))

1 digit number in base u

2 digit number in base then you can represent any number in u

For example, to represent 16 bits we need 4 bits.

13 is D in base 16, 4 bits for each digit???

13 is 31 in base 4, 2 bits 2 digits???

2 digit radix sort of A takes O(n+)

3 digit radix sort O(n+) or O(n+)

…

K digit radix sort O(K(n+))

Convert binary to base 8 just chocking out certain bits and group them

Claim: sorting n elements in the range [], for constant c is O(n) time

Suppose I have a directed graph with n nodes, want to sort the nodes by the product of their in-degrees and out-degrees. How long does it take?

Van Emde Boas Tree (VEB tree)

Maintain a set typically in O(log n) time, say skip list

Insert(k,s)

Del(k,s)

Succ(k,s)

Pred(k,s)

We can do these in log log u time.

How many bits in a number of u ? log u

How many numbers can we have?

T(u)=T()+1 universe of plus a constant time of other things

T(u)=log log u

Suppose I can allocate space u in constant time. Now I have a table of size u

I can’t usually use the memory

I can go to the memory but anything can be there.

How can we allocate a huge amount of memory and safe to use it. We should know whether it’s something I wrote or it’s been there before? Or say whether it’s a safe position to read.

Write in the memory arbitrarily.

Next lecture:

Lower bound of external sorting.

Fusion trees???

# September 22, 2017 Van Emde Boas Tree

Van Emde Boas Tree

Maintain u=[u] set of integers upto u, under

Insert(x,s) O(loglog u)

Delect(x,s) O(logw)

Pred(x,s) min{y|y}

Succ(x,s)

Data payload and link data payload and link

Start out with nothing written in the array.

Log u=w bits

T(u)=T()+1 T’(w)=T’()+1

O(loglog u) O(log w)

First midterm for Van Emde Boas Tree (No Wikipedia allow for notation)

Say High(w)-> high order bits

W.high

Low(w)-> low order bits

W.low

W

W=1001

w.h=10 we shift it to the right and we got higher order

w.low=01 we may reserve order shift it to left and shift it back

Both in constant time

A=

We are not losing bits. It’s all widgets all the way down

Insert(x,w)

Insert(x.low,A[w.high])

Constant time

Succ(x,w) how do we find a successor or a number?

Successor of the low order bit in the widget, or the next non-empty widget

If

Combination Y=<x.high,Succ(x.low,A[x.high])>

The successor of the low order bit in appropriate widget. Keep high order bits, and replace the successor the low order bits. If the successor agrees in the next half of bits.

If it fails, Y not defined, then I need to find Z=next non-empty widget in A , return <Z,succ(Z,w)>

Specified by the minimum thing in there.

Use Van Emde Boas Tree This whole thing W(u) you are allowed to maintain integers up to u

Non empty

Each allows you to maintain integers up to

Insert(x.low,A[x.high])

Insert(x.high,nonEmpty)

But we are only allowed to recurse once. T(u)=T()+1 or it would be too much

How do we maintain it ?

If we insert the first one in Insert(x.high,nonEmpty), we have to store something in that widget with a constant amount of work.

We are going to store a Min and Max

How do you insert into a widget?

Insert(x, w)

If w.min not defined (aka W is empty) //not resursively

w.min=w.max=x return

If w.min=w.max, // not recursively

w.min=Min(x, w.min)

w.max=Max(x, w.max) return

If x<w.min

Swap(x, w.min) // same for w.max

// Guaranteed that x is between min and max

// If (A[x.high].min is defined) we will do it true or not //it’s already not empty, takes constant time

Insert(x.low, A[x.high]) //If we do for first, takes constant time

If(A[x.high].min=A[x.high].max) // this recursion only takes constant time

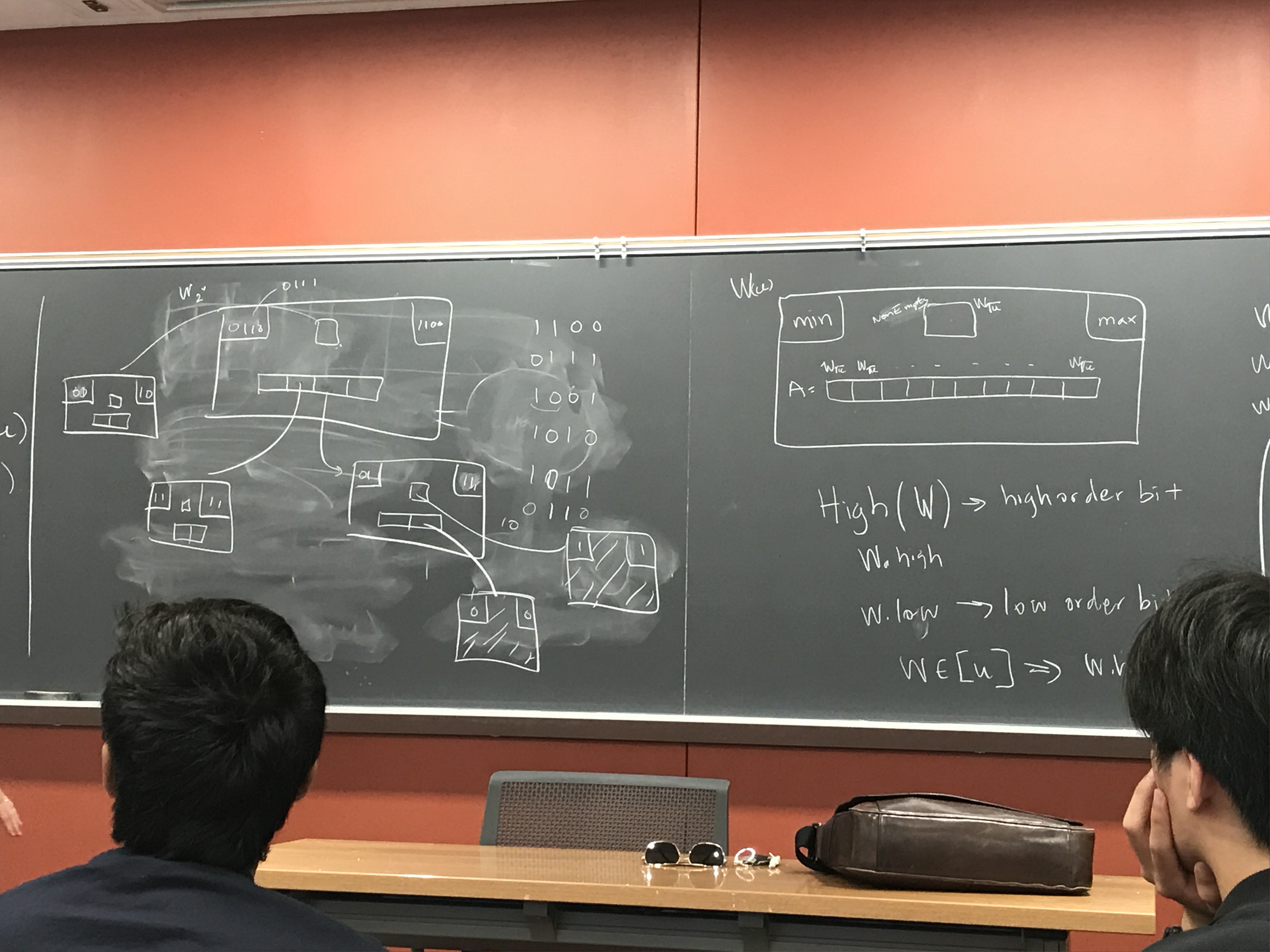
Insert(x.high,nonEmpty)

By keeping min, and max we are only doing one recursion.

We are doing recursively on high orders and and low order bit

Successors use max&min, precede use min

“Data structures are frozen algorithms”



Succ(x,w)

If (x<w.min)

Return w.min

If(x.low < A[x.high].max) //we know we will find a successor

Return <x.high, Succ(x.low,A[x.high])>

Y=Succ(x.high,nonEmpty) // if this barfed, return not defined. That’s what barf for

//Return<Y,A[Y].min>+ error checking

Or: If Y not defined

If x<x.max, return w.max

Else

Barf //a Barf for the recursive call, return not defind

Return<Y,A[Y].min>

Del(x,w)

If x=w.min=w.max, undefined w.min, w.max return

If nonEmpty is empty, fix x.min=x.max is needed

If x=w.min

Y= <nonempty.min, A[nonempty.min].min>

w.min=Y

Del(Y.low,A[nonEmpty.min])

If A[nonEmpty.min] is empty, Del(nonEmpty.min,nonEmpty)

//There would only be one recursive call of Del

If x=w.max, symmetric,

Del(x.low,A[x.high])

if A[x.high] is empty Del(x.high,nonEmpty)

# September 29, 2017

Fusion tree all VEB DPS in O()

VEB tree sorting

Insert each item in a vEMB tree nlogw

Repeated remove min element nlogw/O(nlogw)

A B tree takes I/Os

We pick a node, fetch that node, high branching factor so the tree is much shallower.

The lamest answer is B stands for binary.

X-fast trees (same time, less space, VEB-trees)

Trie out of bits (trie can be built out of a tree)

0111

1101

0110

For every level of the trie, have a pointer to the max and min below it. Build a doubly linked list at the each level. They all point to the last level.

To store these numbers, space:

O(nw) or O(nlogu) w=logu

Query’s next bit is 0

Min is successor

Successor, then left by 1 is predecessor

O(log w) to put and fetch

How long does it take to split a tree

Amortized insertion cost is O(logw+w/w)=O(log w)

Y-fast tree

f: u -> [n]

Pr{f(x)=f(y),x!=y} <= 1/n

If you want to hash n elements, how big does the table have to be so you expect no collisions? Choose k buckets

E[# of collisions] = (each of them )

2-level hash function

We do not expect a collision with quadratic

S has n elements position: K0 K1 ……Kn-1 there are

Ko^2/2 ……Kn-1^2/2 collisions here. Actual number of collisions is summation

f(x)=(ax+b) mod c

The expected # of collisions in the first level =

Summing up collisions in the bucket table.

m/B way merge sort is fastest

integer sorting no one has proved a lower bound.

## Lower bound for external memory sorting

General outline of the proof.

Min number of bits needed to specify answer divided by

Max number of bit is possible to learn by 1 I/O

I/Os of any sorting algorithm

# of orders of input? B words(elements)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

B! for the content of the block. How many bits do we need to distinguish between these guys? For K things, log K.

# of bits needed is log of

Max bits we learn from each I/O?

|  |  |
| --- | --- |
| M-B |  |
|  | B |

Only thing we learn is rank of B elements among those in memory

So we learn bits =O (B log )

Number of I/Os to sort is