CS 173: Discrete Structures

Eric Shaffer
Office Hour: Wed. 12-1, 2215 SC
shaffer1@illinois.edu

Announcements

- Quiz 2 will be returned on Wednesday April 1
- Mid-term 2 in class Wednesday April 8
  - More details later…

Homework 7 clarifications

- Problem 3
  - **procedure** select($a_1, a_2, \ldots, a_n, k$)
    You can assume $1 \le k \le n$
  - **swap**(a_i, a_j)
    means the value at place i in the list is swapped with the value at place j

1, 3, 5
$a_3 = 5$
swap(1, 5)
$a_3 = 1$

a_i & a_j are variables

Geometric Series

- We're interested in finite geometric series

$n+1$ terms

S= a + ar + ar^2 + … + ar^{n-1} + ar^n

$$\sum_{k=0}^{n} ar^k = \frac{a(r^{n+1}-1)}{r-1}$$

$$\sum_{k=m}^{n} ar^k = \sum_{i=0}^{n} ar^i - \sum_{j=0}^{m-1} ar^j$$

What happens if we start the sum from a higher term?

$$= \frac{a(r^{n+1}-1)}{r-1} - \frac{a(r^m-1)}{r-1} = \frac{a(r^{n+1}-r^m)}{r-1}$$

Insertion Sort

Insertion Sort

Input: a unsorted array of real numbers $a_1, a_2, \ldots a_n$, n>1
Output: a sorted array of real numbers $a_1 \le a_2 \le \ldots \le a_n$

```
1.  for j:=2 to n
2.  begin
3.      i:=1
4.      while a_j>a_i
5.          i:=i+1
6.      m := a_j
7.      for k := 0 to j-i-1
8.          a_{j-k} := a_{j-k-1}
9.      a_i:=m
10. end
```

$O(n^2)$

assignments: 2
one comparison each iteration: 3
4

$\Rightarrow \sum_{j=2}^{n} j = O(n^2)$

4, 3, 2, 1

j=2: 3, 4, 2, 1

j=3: 2, 3, 4, 1

j=4: 1, 2, 3, 4

Recursive Binary Search

BinarySearch(x,i,j, $a_1,a_2,\ldots,a_n$)
Input: a sorted array A of **n** numbers and a ~~number x~~
Output: location of x in A

```
1.  if (n > 1)
2.     m := ⌊(i + j)/2⌋
3.     if x > a_m then BinarySearch(x, m+1, j, a_1, a_2 .., a_n)
4.     else BinarySearch(x, i, m, a_1, ... a_n)

5.  else
6.     if x = a_i then output:=I {x is at position i}
7.     else output:=0 {x is not in list}
```

x value searching for

$a_1, \ldots, a_n$

i, j current search range

Cwork

Base

Recursive Binary Search

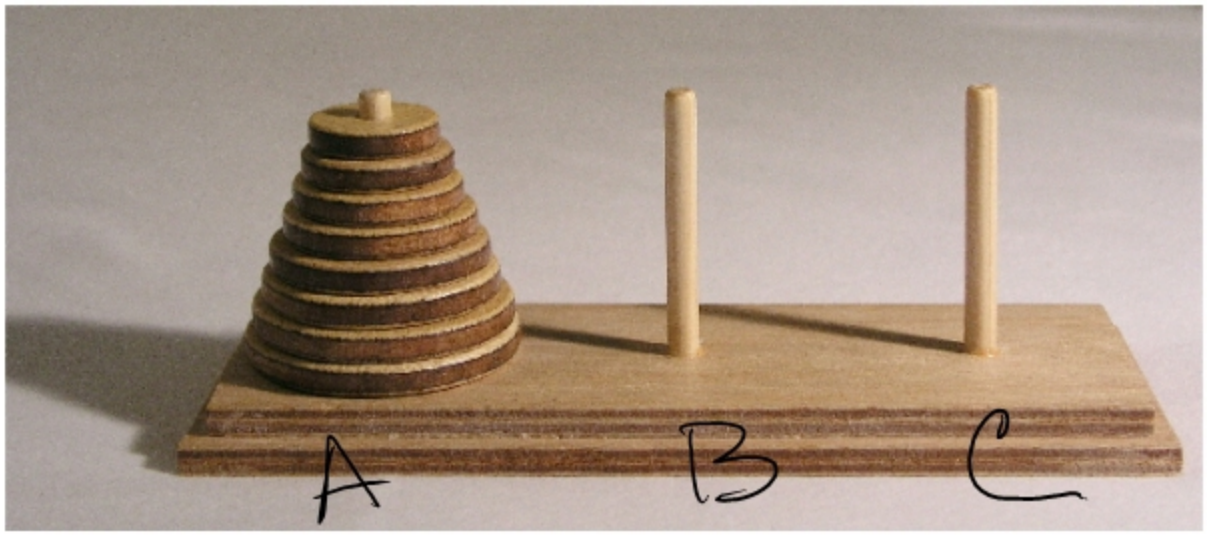
An appropriate recurrence relation for binary search is:

A. $T(n) = C + T(n-1)$
B. $T(n) = C + T(n/2)$
C. $T(n) = n + T(n/2)$
D. Depends on the data.

Base case: $T(1) = O(1)$

Tower of Hanoi…

The Tower of Hanoi invented by French mathematician, Edouard Lucas, in 1883.



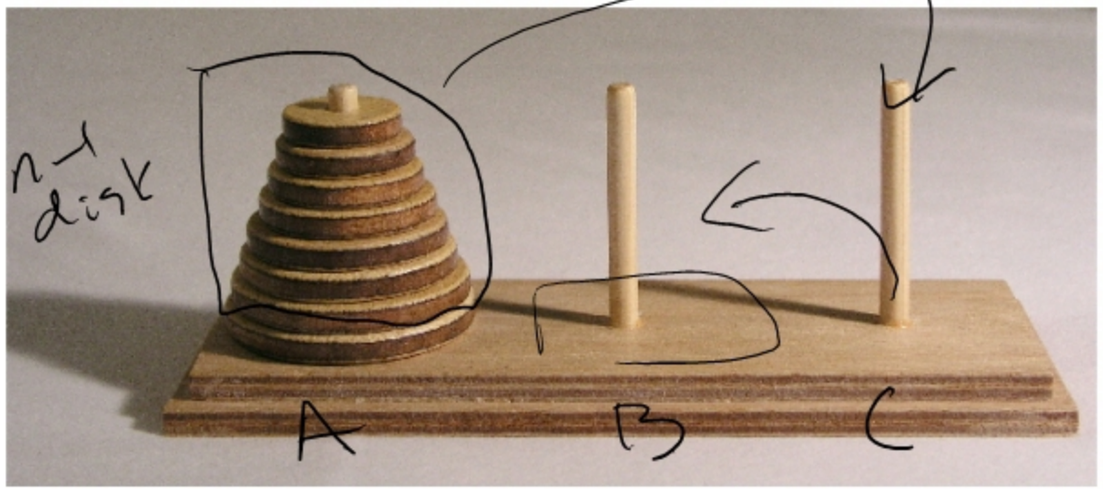
Move disks from post A to post B without putting larger over smaller.

Animation:

http://www.mazeworks.com/hanoi/

Tower of Hanoi…

The Tower of Hanoi invented by French mathematician, Edouard Lucas, in 1883.



TH(A, B, C, n)
start post
end post
disks

```
TH (A, B, C,n)
1. If n = 1 then Move(A,B)
2. else
3. begin
4.     TH(A,C,B,n-1)
5.     Move(A,B)
6.     TH(C,B,A,n-1)
7. end
```

Base

Towers of Hanoi efficiency (or lack thereof)

```
TH (A, B, C,n)
1.    If n = 1 then Move(A,B)
2.    else
3.    begin
4.       TH(A,C,B,n-1)
5.       Move(A,B)
6.       TH(C,B,A,n-1)
7.    end
```

$T(1) = O(1)$

$T(n) = 2T(n-1) + C$

Now let's analyze the running time. $T(1) = C$

$T(n) = 2\,T(n-1) + C$

$= 2\,(2\,T(n-2) + C) + C = 4\,T(n-2) + 3C$

$= 4\,(2\,T(n-3) + C) + 3C = 8\,T(n-3) + 7C$

$\ldots = 2^k\,T(n-k) + (2^k-1)C$

What is k when this bottoms out at 1?

$k = n-1$

$= 2^{n-1}\,T(1) + (2^{n-1}-1)C = (2^n-1)\,C$

$= O(2^n)$

$(2^{n-1})\,c + (2^{n-1}-1)\,c$

$= c\,(2 \cdot 2^{n-1} - 1) = c\,(2^n - 1)$

Does it work…?

Is the algorithm correct?

Does it do the right thing for 1 disk?

Assume it does the right thing for n-1 disks. (IH)

And finally, it DOES do the right thing for n disks:
move n-1 out of the way, move the biggest disk, move n-1 back.

Recursive sorting....

MergeSort ($a_1, a_2, \ldots, a_n$)

```
1. If n=1 then
2.   {return the list of size 1, it's sorted}
3. Else
4.     m:= ⌊n/2⌋
5.     Merge(MergeSort(a_1,…,a_m),MergeSort(a_{m+1},a_n))
```

Base case

sort n/2 items

sort n/2 items

merge 2 sorted lists into a single sorted list

Recursive sorting....

MergeSort ($L = a_1, a_2, .., a_n$) {L is a list}

```
1. If n=1 then return L
2.   {return the list of size 1, it's sorted}
3. Else
4.     m:= ⌊n/2⌋
5.     Merge(MergeSort(a_1,…,a_m),MergeSort(a_{m+1},a_n))
```

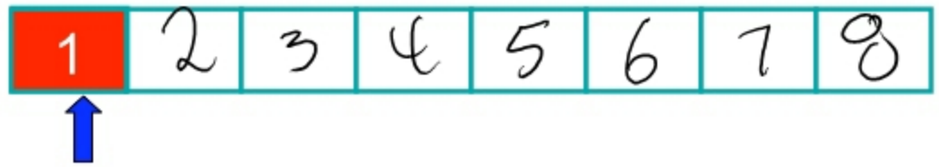
Merge(L1,L2) {two sorted lists $L1=a_1, a_2, .., a_u$ $L2=b_1, b_2, .., b_v$}

```
1. L = empty list
2. While (L1 and L2 are non-empty)
       1. Remove smaller of a_1 and b_1 and append it to L
3. If L1 or L2 is non-empty, append it to L
```

Merging takes O(n) time...

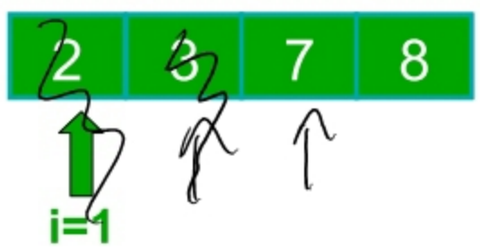
size of L: |L| = n

L:

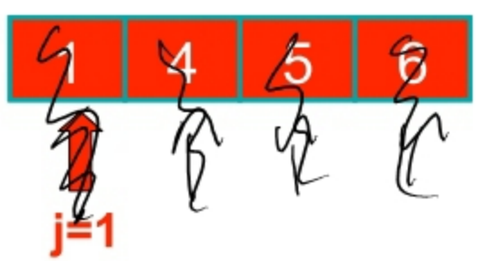


|L| = |L1| + |L2|

L1:



L2:



Total work

O(n)

O(n) assignments

So |L1| = O(n)

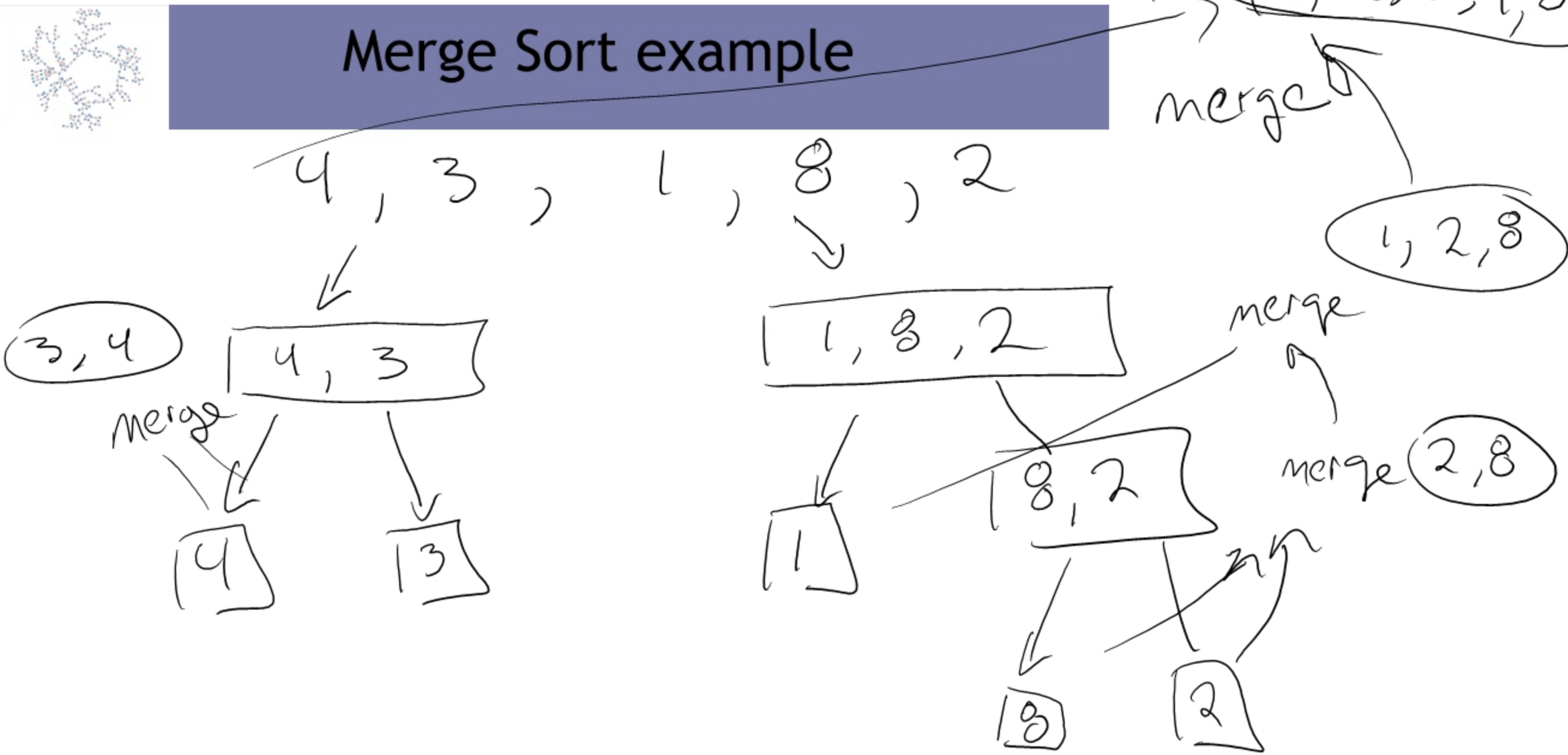
|L2| = O(n)

Max # compares

= |L1| + |L2| - 1

= O(n)

Merge Sort example



Recursive sorting….

Animations:

http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/mergeSort/mergeSort.html

http://www.sorting-algorithms.com/

Merge sort efficiency

Write down the recurrence relation for T(n)

-- number of operation to merge sort n items

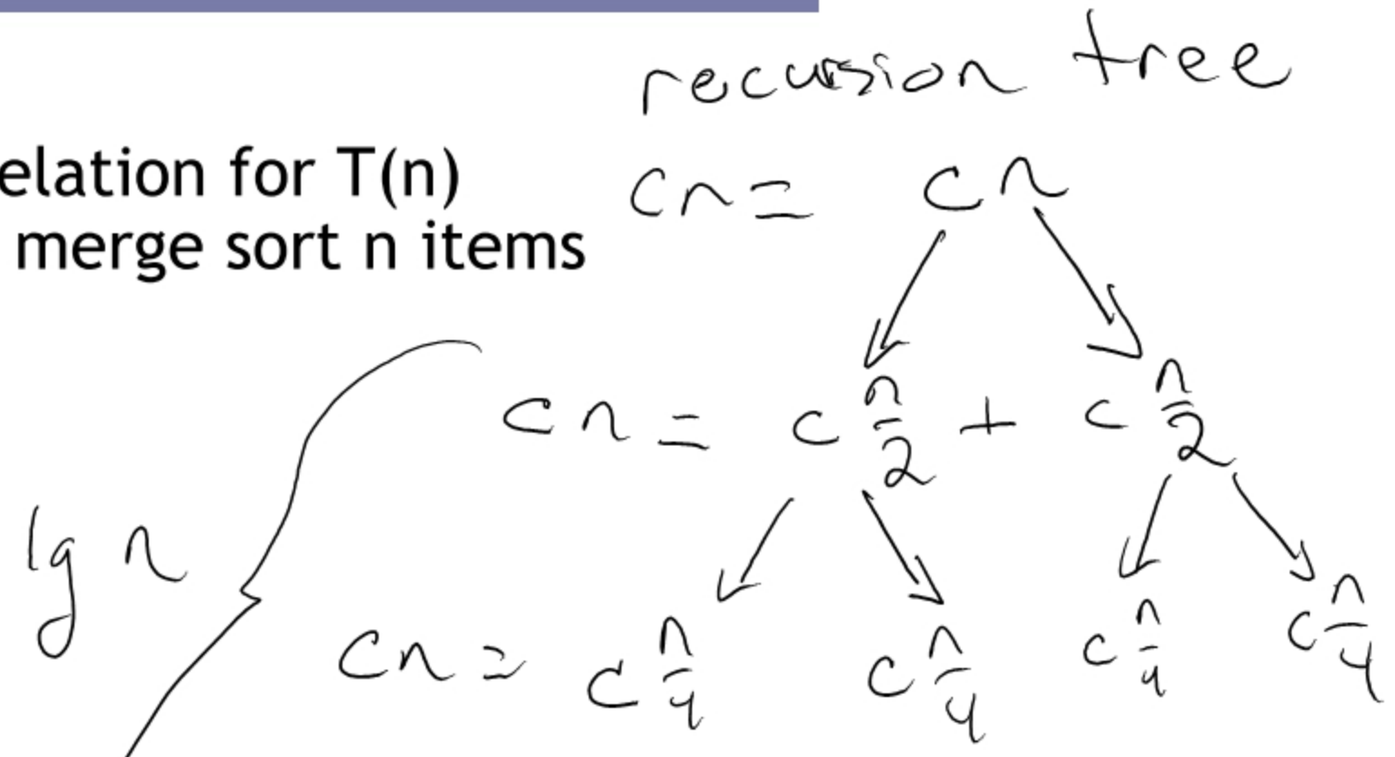
-- worst-case analysis

What's the base case?

$T(1) = c$

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

recursion tree



$T(n) = (\lg n + 1)cn$

$= cn\lg n + cn$

$= O(n \lg n)$