Analysis of Algorithms - CS 323
Lecture #4 - February 24, 2016

Notes by: Gurpreet Singh

1. Using the formal definition of Ω, prove that $3n^3 + 2n^2 + n = \Omega(1000n^2 + 2000n + 3000)$.

   N_0=1 and c = .00001

   The definition of Omega is f(n)>=cg(n) where c is constant that makes this true, and n_0 is a integer.

2. Using the formal definition of Θ, prove that 100nlogn + 50n = Θ(50nlogn + 100n).

   C_2(50nlogn)<=100nlogn+50n<=c_1(50nlogn+100n)

   C_2=.5 n_0=1 c_1=2

3. For each, state whether it is true or false and offer a brief explanation:
   a. if f(n) = O(g(n)) and g(n) = O(f(n)), then f(n) = g(n)**False**
   b. if f(n) = Θ(g(n)) and g(n) = Θ(h(n)), then f(n) = Θ(g(n))**True**
   c. if f(n) = O(g(n)) and g(n) = O(h(n)), then h(n) = Ω(f(n))**True**
   d. if f(n) = ω(g(n)) and f(n) = o(h(n)), then h(n) = Θ(g(n))**False**
   e. $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) \times f_2(n) = O(g_1(n) \times g_2(n))$**True**

4. Suppose you want to find both the minimum *and* maximum of an array of integers. How might you combine the two separate algorithms for minimum and maximum into one algorithm that is more efficient? What would be the best, average, and worst case of your combined algorithm?

```
If a[i]<tempMin
    tempMin=a[i]
else if a[i] > tempMax
    tempMax=a[i]
```

   Best=n-1 because you only do one comparsion

   Worest=2n-1 because you enter both if statements

   Average = 2n/2 because you enter half one if and other if

5. As observed in class, if after one pass (iteration) through the array in Bubble Sort there are no swaps, then the array is already sorted. Modify the Bubble Sort algorithm from class to take advantage of this observation. What is the new best, average, and worst case analysis of the modified algorithm?

6. One problem with Bubble Sort is that while large elements move quickly to their final position toward the end of the array, small elements move only slowly to their final position toward the beginning of the array. This phenomenon has been dubbed "Rabbits and Turtles" (since the hare is fast and the tortoise is slow.) Design a variation of Bubble Sort that uses a back-and-forth

approach to ensure that both are achieved at equal speed. (Hint: see https://en.wikipedia.org/wiki/Bubble_sort#Rabbits_and_turtles.)

Look at Shaker Sort

Best = n-1 because the list is already a sorted list and the algorithm only goes throw it once

Average= 1/n!(probability for each literation)

Worest=n(n-1)/2 this is the answer because the algorithm will do multiple swaps and it will do multiple swaps at every comparison

7. Explain why T(1) = 0, T(n) = T(n-1) + n-1 aptly describes the number of comparisons for Bubble Sort and then, using the techniques from class, solve the recurrence.

T(n) - ~~T(n-1)~~=n-1 this is telescoping because we expand
~~T(n-1)~~- T(n-2)=n-2 the recurrence until T(1) and all almost
. everything is cancelled until T(n) – T(1)
. is left but if we add the left side we get
T(2) - T(1) =1 n(n-1)/2 and the right we have T(n)

T(n)=n(n-1)/2

8. Write pseudocode for an algorithm (can be made up) whose time complexity is described by the recurrence T(1) = 1, T(n) = T(n/2) + n.

```
myProgram(n)

        if n>1
          myProgram(n/2)          this line of code reduced the list
                                  to T(n/2).
        for i=1 to n{             this for loop is complity of n
        print(~~)
}                                 this is same as T(n/2) + n
```

9. Solve the recurrence in the previous question.

$n=2^k$

S(k)=T(n)

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

$S_k = S(k-1) = 2^k$

$S(k) - \cancel{S(k-1)} = S^k$

$\cancel{S(k-1)} - S(k-2) = S^{k-1}$

.

.

$S(1) - S(0) = 2$

LeftSide $= S(k) - S(0)$

RightSide $= 2^{k+1} - 1 - 1 = 2^{k+1} - 2$

$S(k) = 2^{k+1} - 1$

$T(n) = 2^{\lg n + 1}$

$= 2^{\lg n} * 2 - 1$

$= 2n - 1$

Lecture

SELECTION SORT

```
    SelectionSort(int[] arr){

   for (int i = 0; i < arr.length - 1; i++)
   {
      int index = i;
      for (int j = i + 1; j < arr.length; j++)
         if (arr[j] < arr[index])
            index = j;

      int smallerNumber = arr[index];
      arr[index] = arr[i];
      arr[i] = smallerNumber;
   }
   return arr;
}
```

n-1 comparison => n(n-1)/2 (worst, average , Best)
1 swap => n-1(worst)

This a really good algorithm for a small list but as the list or the data gets bigger the algorithm behaves like $O(n^2)$. The reason is because there are two FOR loops and this is a time consuming.

INSERTION SORT

Sorted	Unsorted

```
 for (int i = 1; i < a.length; i++){

        int temp = a[i]
        int j;
        for (j=I -1; j >= 0; && temp < a[j]; j--){

                a[j+1]=a[j]
                a[j+1]=temp
        }
}
```

Worst = (n(n-1))/2
Average = (n(n-1))/2
Best = n-1

This is slow algorithm because there are two for loops. The List has to go through both loops with can a lot of time consuming.

TWIST TO INSERTION SORT

-Use binary search to find where to insert, than Bulk move the insertion point to empty point
-Sentinel
Adding the smallest element to negative index or before the first index.

SHELL SORT

Worst = n^2

Best = $n^{1.5}$

MERGE SORT (Divide and Conquer)

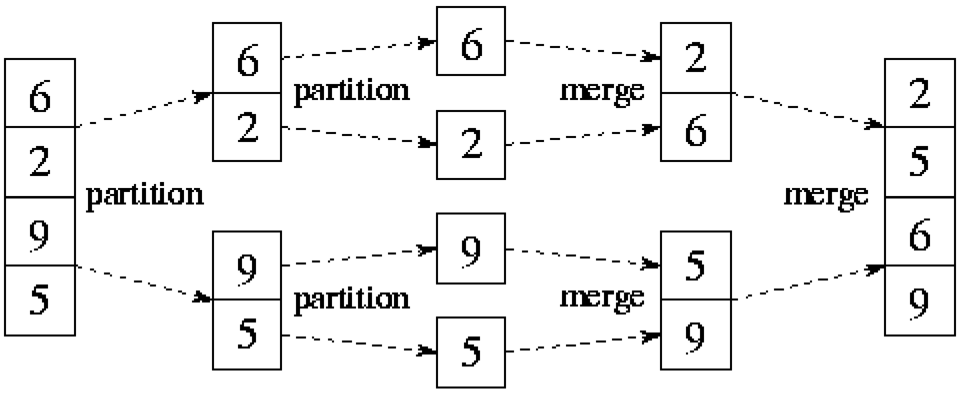
```
merge(Comparable[ ] a, Comparable[ ] tmp, int left, int right, int rightEnd )
   {
      int leftEnd = right - 1;
      int k = left;
      int num = rightEnd - left + 1;

      while(left <= leftEnd && right <= rightEnd)
         if(a[left].compareTo(a[right]) <= 0)
            tmp[k++] = a[left++];
         else
            tmp[k++] = a[right++];

      while(left <= leftEnd)    // Copy rest of first half
         tmp[k++] = a[left++];

      while(right <= rightEnd)  // Copy rest of right half
         tmp[k++] = a[right++];

      // Copy tmp back
      for(int i = 0; i < num; i++, rightEnd--)
         a[rightEnd] = tmp[rightEnd];
   }
}
```



This is algorithm that divides and keeps on dividing until its left with a single element. Then it compares the element with one another and then

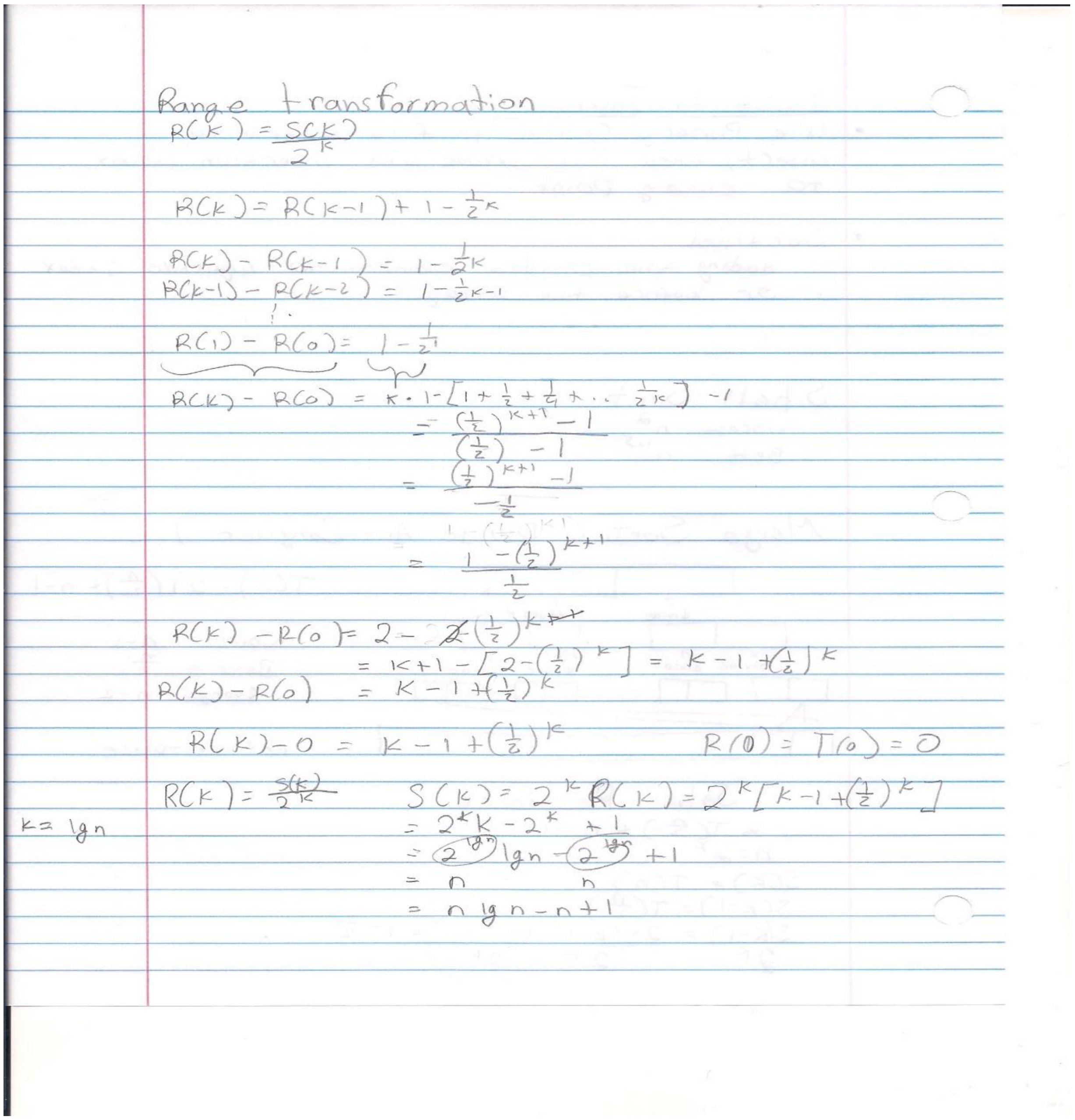
remerges the sorted list with other sorted list. This keeps in happing until the list is sorted.

There is only one comparison at the end of the division. And also when the lists are merging.

Solve Recurrence

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

$n=2^k$
$S(k) = T(n)$
$S(k-1) = T(n/2)$
$S(k-1)/2 = sS(k-1)/2^k + (1-1/2^k)$
Range Transformation



Range transformation

$$R(K) = \frac{S(K)}{2^K}$$

$$R(K) = R(K-1) + 1 - \frac{1}{2^K}$$

$$R(K) - R(K-1) = 1 - \frac{1}{2^K}$$
$$R(K-1) - R(K-2) = 1 - \frac{1}{2^{K-1}}$$
$$\vdots$$
$$\underbrace{R(1) - R(0)} = \underbrace{1 - \frac{1}{2^1}}$$

$$R(K) - R(0) = K \cdot 1 - \left[1 + \frac{1}{2} + \frac{1}{4} + \dots \frac{1}{2^K}\right] - 1$$
$$= \frac{\left(\frac{1}{2}\right)^{K+1} - 1}{\left(\frac{1}{2}\right) - 1}$$
$$= \frac{\left(\frac{1}{2}\right)^{K+1} - 1}{-\frac{1}{2}}$$
$$= \frac{1 - \left(\frac{1}{2}\right)^{K+1}}{\frac{1}{2}}$$

$$R(K) - R(0) = 2 - \cancel{2}\left(\frac{1}{2}\right)^{K ~~+1~~}$$
$$= K+1 - \left[2 - \left(\frac{1}{2}\right)^K\right] = K - 1 + \left(\frac{1}{2}\right)^K$$
$$R(K) - R(0) = K - 1 + \left(\frac{1}{2}\right)^K$$

$$R(K) - 0 = K - 1 + \left(\frac{1}{2}\right)^K \qquad R(0) = T(0) = 0$$

$$R(K) = \frac{S(K)}{2^K} \qquad S(K) = 2^K R(K) = 2^K\left[K - 1 + \left(\frac{1}{2}\right)^K\right]$$

$K = \lg n$

$$= 2^K K - 2^K + 1$$
$$= \underbrace{2^{\lg n}}_{n} \lg n - \underbrace{2^{\lg n}}_{n} + 1$$
$$= n \lg n - n + 1$$

QUICK SORT

<x	X	>x

Picking X

1)first/last
2)random

3) median of 3

1) Picks from unsorted list then scans the sorted list, while doing that we move element to right. But before that we need to pick a pivot point
2) There are various ways of picking that number. The best method is picking the median of the last and first and there average.
3) Then you compare the pivot point with the unsorted list and if its less than, than move the number to left

Worst = T(n)=T(n-1) + n-1 = (n(n-1))/2
Best   = T(n) = T(n/2) + (n/2) + T(n/2) + n -1 = nlogn
Average = Next time in class