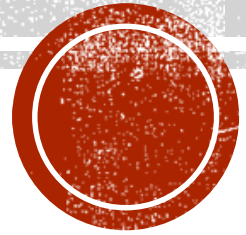
M2: C# QUICK REVIEW & SEARCHING ALGORITHMS SORTING ALGORITHMS

Summer 2019 – CSC 395 – ST: Algorithm & Data Structure Concepts



SOME RESOURCES

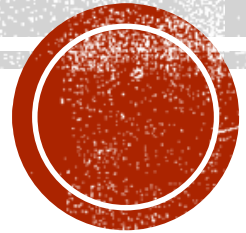
- https://legacy.gitbook.com/book/cathyatseneca/data-structures-and-algorithms/details

- See chapter 9 (and section 3.1) from http://people.cs.vt.edu/~shaffer/Book/C++3e20130328.pdf

- For test #1 be able to implement on paper the searching & sorting algorithms (and other) we'll see next

C# QUICK REVIEW

Summer 2019 – CSC 395 – ST: Algorithm & Data Structure Concepts



“HELLO WORLD!”

- Use Ctrl+F5

  rather than ending code with:

  ```
  Console.ReadKey();
  ```

```
using System;

namespace HelloWorldApplication {
   class HelloWorld {
      static void Main(string[] args) {
         /* my first program in C# */
         Console.WriteLine("Hello World");
         Console.ReadKey();
      }
   }
}
```

YOU NEED TO MASTER LOOPS IN HERE!

- Example: Let's write a C# program that will display every character from a given string…
- Example: Let's write a C# program that will display every second character from a given string…
- Example: Let's write a C# program that will count and display the number of times the letter C appears in a user-given input.
- Example: How do you compare integers? What about strings?

YOU NEED TO MASTER LOOPS IN HERE!

- Please review **references**:
  - What happens when we pass by value? For example:
  - What happens when we pass by reference?
  - What happens when we pass an array?

```
void f(int x){x = 10;}
void f(ref int x){x = 10;}
void f(int[] arr){arr[0] = 10;}
```

- Example: Swap values, return a value without return …

YOU NEED TO MASTER LOOPS IN HERE!

- Example: Write a program that will count the number of characters and number of lines from a given file, say newspaper.txt

- To read from files one could use:
  - string[] arr1 = System.IO.File.ReadAllLines("newspaper.txt"); //creates array arr1

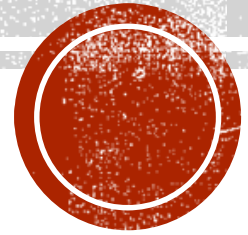
NOTE: EXTRA POINTS CHALLENGES

Look at the end of this file (slides: Interview PROBLEMS and Programming COMPETITION) for problems that carry extra points opportunities.

- Each problem will specify a number of extra points available.
  - E.g. 10 extra points available
- These extra points will be added to a(any) homework grade
- It's all or nothing: you need to completely solve a particular problem in order to get the extra credit associated with it
- You will have to come to my office and explain me your code, and why it works.
  - Please email me your solution(s) and schedule an appointment
- Deadline to submit them: usually the same as the homework.

SEARCHING ALGORITHMS

Summer 2019 – CSC 395 – ST: Algorithm & Data Structure Concepts



INTRO

- If I ask you to search for the word "Saint" in a book how would you look for it?
  - Case 1: looking into any book (type of problem: **unsorted array**)
    - Typical algorithm: **Sequential search**
  - Case 2: looking into a dictionary (type of problem: **sorted array**)
    - Typical algorithm: **Binary search**

SEQUENTIAL SEARCHING

- Sequential search is used when the items in the list are in random order.

- **Basic idea**: Begin at the beginning of a set of records and move through each record until you find the record you are looking for or you come to the end of the records.

- **Algorithm**: Start at the beginning of the array and compare each accessed array element to the value you're searching for.
  - If you find a match, the search is over.
    Return it's index.

  - If you get to the end of the array without generating a match, then the value is not in the array.
    Return -1.

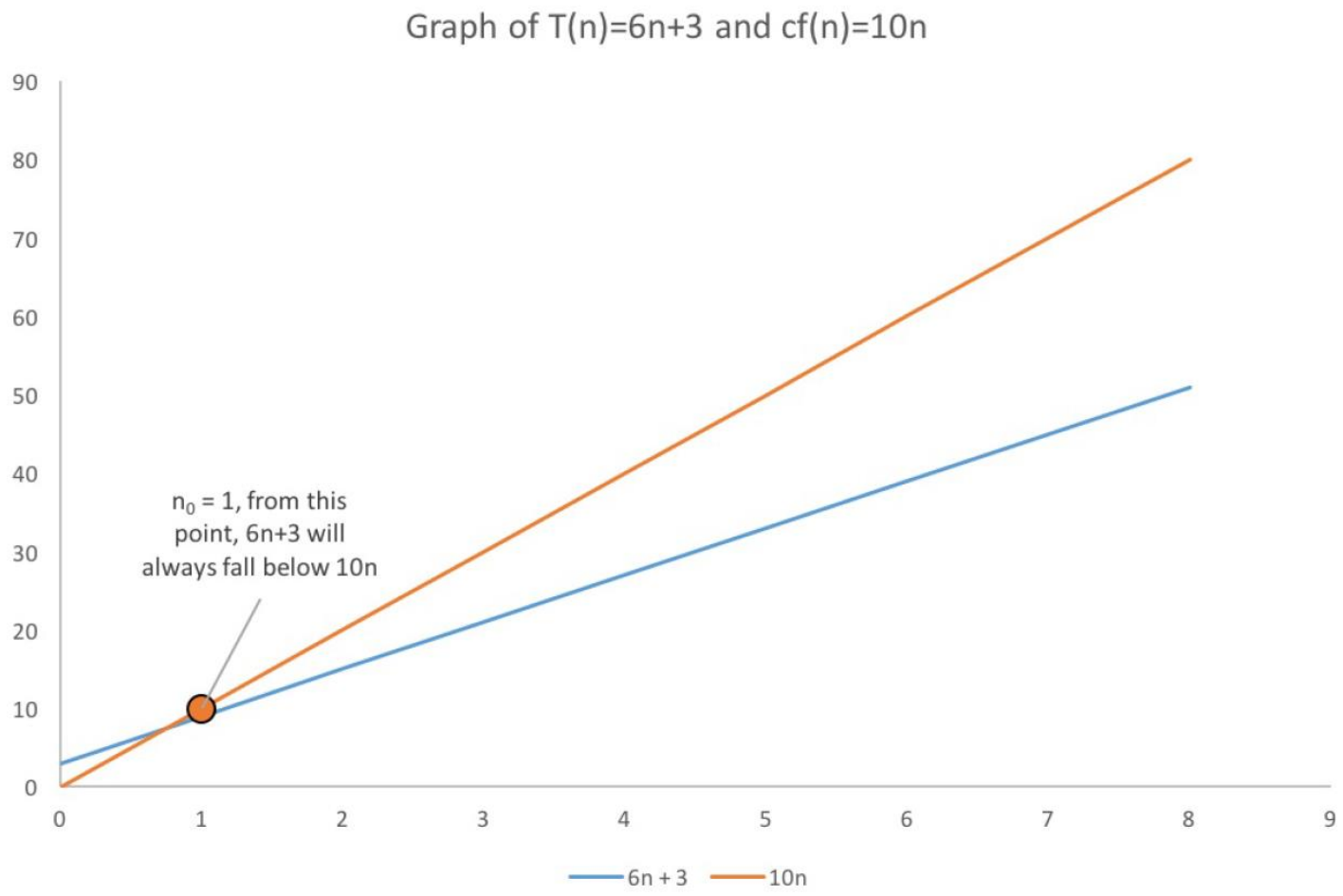
SEQUENTIAL SEARCHING - IMPLEMENTATION

- Coding in C# …
  - ```
    static int SeqSearch(int[] arr, int key)
    {
        int ret = -1;
        for(int index=0; index<arr.Length && ret==-1; index++)
                if(arr[index]==key)
                        ret= index;
        return ret;
    }
    ```

- What is the running time(the big oh)? What about
  - Assuming arr.Length runs in constant time, count the number of operations
  - Graph to compare 6n+1 with 10n … hence we obtain O(n)


    ```

SEQUENTIAL SEARCHING



SEARCHING FOR MIN AND MAX VALUES

- Search an array (or other data structure) for **minimum** and **maximum** values.
  - In an ordered array, searching for these values is a trivial task. So we'll assume it not sorted
    - How would you find the largest number in a 100-page book?

  - Assign the first element of the array to a variable as the minimum value.
  - Begin looping through the array, comparing each successive array element with the minimum value variable.
  - If the currently accessed array element is less than the minimum value, assign this element to the minimum value variable.
  - Continue until the last array element is accessed.
  - The minimum value is stored in the variable

MINIMUM VALUE SEARCH - IMPLEMENTATION

```
static int FindMin(int[] arr)
{
      int minVal=arr[0];
      for(int i=0;i<arr.length;i++)
            if(arr[i]<minVal)
                  minVal=arr[i];
      return minVal;
}
```

- How would you change this to search for Max value?
- What is the running time of this algorithm? Big oh, big omega, big theta?
- Change this algorithm to return position, rather than value…

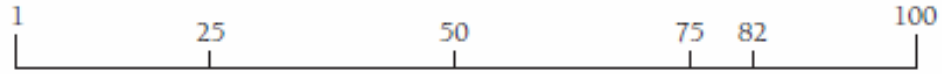
GUESSING GAME

- Let's implement the guessing game. The computer will randomly choose a number between 1 and 100. You are trying to guess it. For every guess the computer would tell you if your guess is correct, too low, or too high. What is your strategy?
- The best strategy then is to choose 50 as the first guess.
  - If that guess is too high, you should then guess 25.
  - If 50 is to low, you should guess 75.
- Each time you guess, you select a new midpoint by adjusting the lower range or the upper range of the numbers (depending on if your guess is too high or too low), which becomes your next guess.
- As long as you follow that strategy, you will eventually guess the correct number.

GUESSING GAME (2)

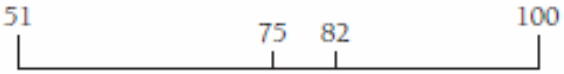
- One of you please select a secret number between 1 and 100.
  - Let's see this search

Guessing Game-Secret number is 82



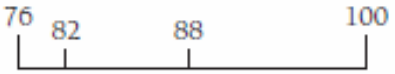
First Guess : 50

Answer : Too low



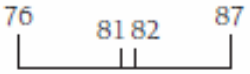
Second Guess : 75

Answer : Too low



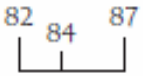
Third Guess : 88

Answer : Too high



Fourth Guess : 81

Answer : Too low



Fifth Guess : 84

Answer : Too high

Midpoint is 82.5, which is rounded to 82

Sixth Guess : 82

Answer : Correct

BINARY SEARCH BY EXAMPLE

- Let's create a sorted list of 16 numbers.
  - Then let's search for an existing one using binary search. How many steps did it take you?
  - Then let's search for one that is not in the list using binary search. How many steps it took?

  - Do the same thing using sequential search …

BINARY SEARCH - ALGORITHM

- **For a binary search to work the data must be sorted**. In this case we assume that the data is sorted from smallest (at arr[0]) to biggest (at arr[size-1]).

- Set the lower and upper bounds of the search.
  - At the beginning of the search, this means the lower and upper bounds of the array.
- Calculate the midpoint of the array.
  - The array element stored at this position is compared to the searched-for value.
    - If they are the same, the value has been found and the algorithm stops.
    - If the searched-for value is less than the midpoint value, then look into the first half of searched **sub**list
      - a new upper bound is calculated by subtracting 1 from the midpoint.
    - If the searched-for value is greater than the midpoint value, then look into the second half of searched **sub**list
      - a new lower bound is calculated by adding 1 to the midpoint.
  - The algorithm iterates until the lower bound is greater than the upper bound, which indicates the array has been completely searched.
    - If this occurs, a `-1` is returned, indicating that no element in the array holds the value being searched for.

BINARY SEARCH - IMPLEMENTATION

```
static int binSearch(int[] arr, int key){
{
      int ret=-1;

      int low=0;
      int high=arr.Length-1;
      int mid;

      while(low<=high && ret==-1)
      {
            mid=(low+high)/2;
            if(arr[mid] == key)
                  ret=mid;
            else if(arr[mid] < key)
                  low= mid+1;
            else
                  high=mid-1;
      }
      return rc;
}
```

BINARY SEARCH - RECURSIVE

```
static int recBinSearchHelp(int[] arr, int key,int low, int high){
{
      if(low>high)
            return -1;

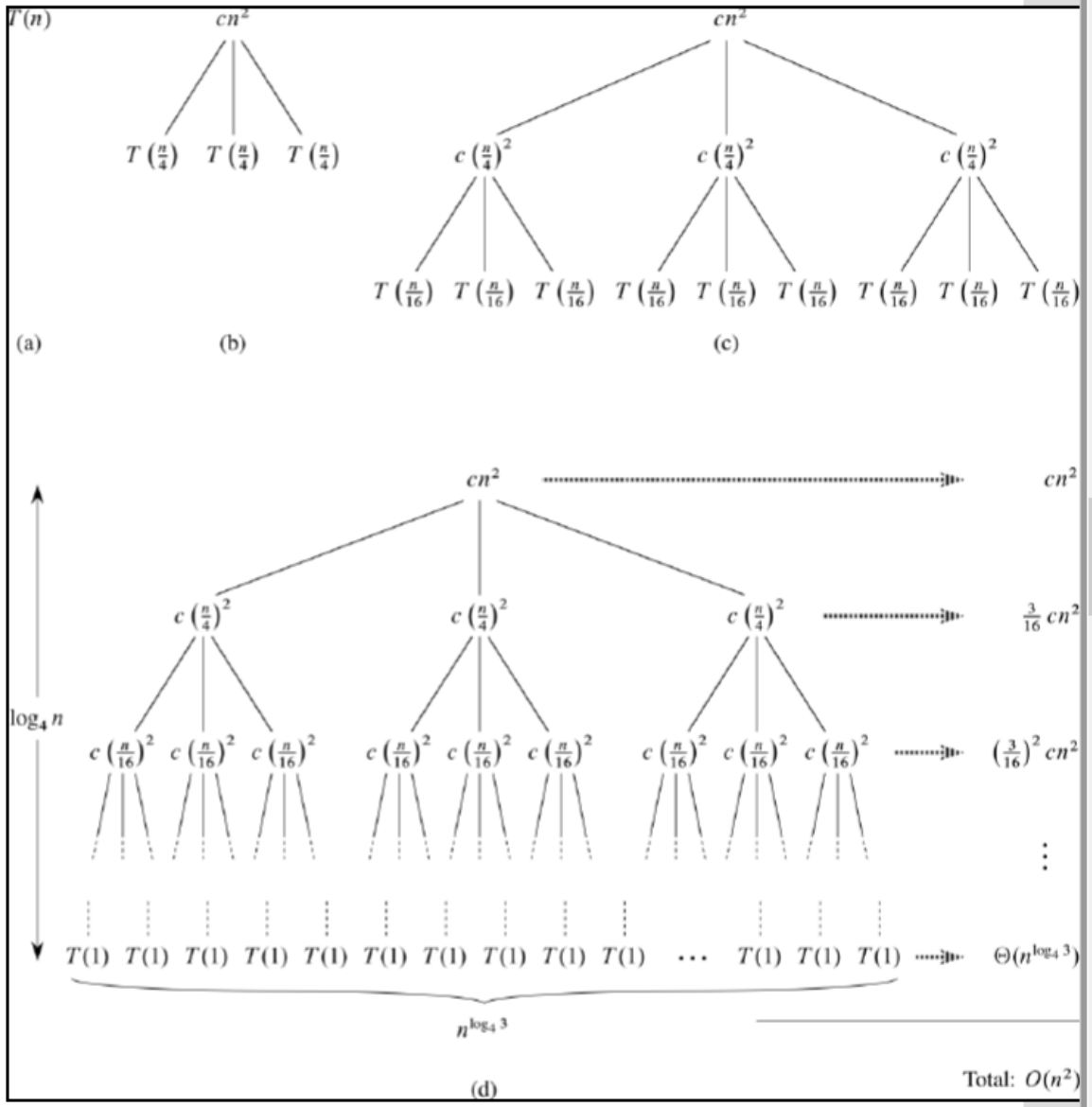
      int mid=(low+high)/2;

      if(arr[mid] == key)
            return mid;
      else if(arr[mid] < key)
            return recBinSearchHelp(arr, key, mid+1,high);
      else
             return recBinSearchHelp(arr, key, low,mid-1);
}
```

```
static int recBinSearch(int[] arr, int key)
{
      return recBinSearchHelp(arr, key,0, arr.Length-1)
}
```

THE RECURSION TREE METHOD

- In a ***recursion tree***, each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.
- We sum the costs within each level of the tree to obtain a set of per-level costs, and then we sum all the per-level costs to determine the total cost of all levels of the recursion.
- Recursion trees are particularly useful when the recurrence describes the running time of a divide-and-conquer algorithm.
- A recursion tree is best used to generate a good guess, which is then verified by the substitution method
- When using induction, you can often tolerate a small amount of "sloppiness"
- For example, $T(n) = 3\,T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$ can we written out as $T(n) = 3\,T(n/4) + cn^2$



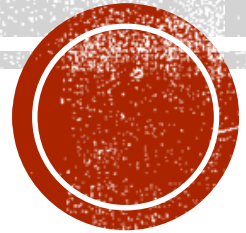
BINARY SEARCH – RUNNING TIME

- What is the running time?

- How many times can you divide n by 2? That is 2 to what power fits into n?
  - $T(n)=T(n/2)+c= T(n/2^2)+2c=T(n/2^3)+3c=\ldots=T(n/2^x)+ x*c$
  - $1 = n / 2^x$
  - $2^x = n$
  - $\log_2(2^x) \quad = \log_2 n$
  - $x * \log_2(2) = \log_2 n$
  - $x * 1 \qquad = \log_2 n$
- Or, using the master theorem: T(N) = T(N/2) + O(1) // the recurrence relation
  - Here, a = 1, b = 2 => log (a base b) = 1
  - also, here f(N) = n^c log^k(n) //k = 0 & c = log (a base b)
  - So, T(N) = O(N^c log^(k+1)N) = O(log(N))

SORTING ALGORITHMS

Summer 2019 – CSC 395 – ST: Algorithm & Data Structure Concepts



SOME RESOURCES

- https://legacy.gitbook.com/book/cathyatseneca/data-structures-and-algorithms/details

- See chapter 7 from http://people.cs.vt.edu/~shaffer/Book/C++3e20130328.pdf

OTHER USEFUL LINKS

- visualization:
  - Bubble sort: https://visualgo.net/en/sorting

  - Bubble sort - cathy at seneca: http://cathyatseneca.github.io/DSAnim/web/bubble.html
  - Selection sort - cathy at seneca: http://cathyatseneca.github.io/DSAnim/web/selection.html
  - insertion sort - cathy at seneca: http://cathyatseneca.github.io/DSAnim/web/insertion.html
  - merge sort - cathy at seneca: http://cathyatseneca.github.io/DSAnim/web/merge.html
  - quick sort - cathy at seneca: http://cathyatseneca.github.io/DSAnim/web/quick.html
  - bubble sort - liang: http://www.cs.armstrong.edu/liang/animation/web/BubbleSort.html

  - selection sort - liang: http://www.cs.armstrong.edu/liang/animation/web/SelectionSort.html
  - insertion sort - liang: http://www.cs.armstrong.edu/liang/animation/web/InsertionSort.html
  - merge sort - liang: http://www.cs.armstrong.edu/liang/animation/web/MergeList.html
  - quick sort - liang: http://www.cs.armstrong.edu/liang/animation/web/QuickSortPartition.html

  - EXTRA: liang radix sort: http://www.cs.armstrong.edu/liang/animation/web/RadixSort.html

OTHER USEFUL LINKS(2)

- videos:
  - Bubble Sort: https://www.youtube.com/watch?v=6Gv8vg0kcHc
  - Bubble Sort: https://www.youtube.com/watch?v=yIQuKSwPlro
  - Selection Sort: https://www.youtube.com/watch?v=6nDMgr0-Yyo
  - Selection Sort: https://www.youtube.com/watch?v=f8hXR_Hvybo
  - Insertion Sort: https://www.youtube.com/watch?v=c4BRHC7kTaQ&list=PLAB1DA9F452D9466C&index=2
  - Insertion Sort: https://www.youtube.com/watch?v=DFG-XuyPYUQ
  - skipped ~~shell sort~~: https://www.youtube.com/watch?v=wdrwwl5Ib9g
  - quick sort in 4 minutes: https://www.youtube.com/watch?v=Hoixgm4-P4M

INTRO

- Sorting is also one of the most frequently performed computing tasks.
  - We might sort the records in a database so that we can search the collection efficiently. We might sort the records by zip code so that we can print and mail them more cheaply.

- We will consider several algorithms that can sort data. The performance of these algorithms can vary widely.
  - Some algorithms are complex but work fast. Some are slow but very easy to write.

BUBBLE SORT

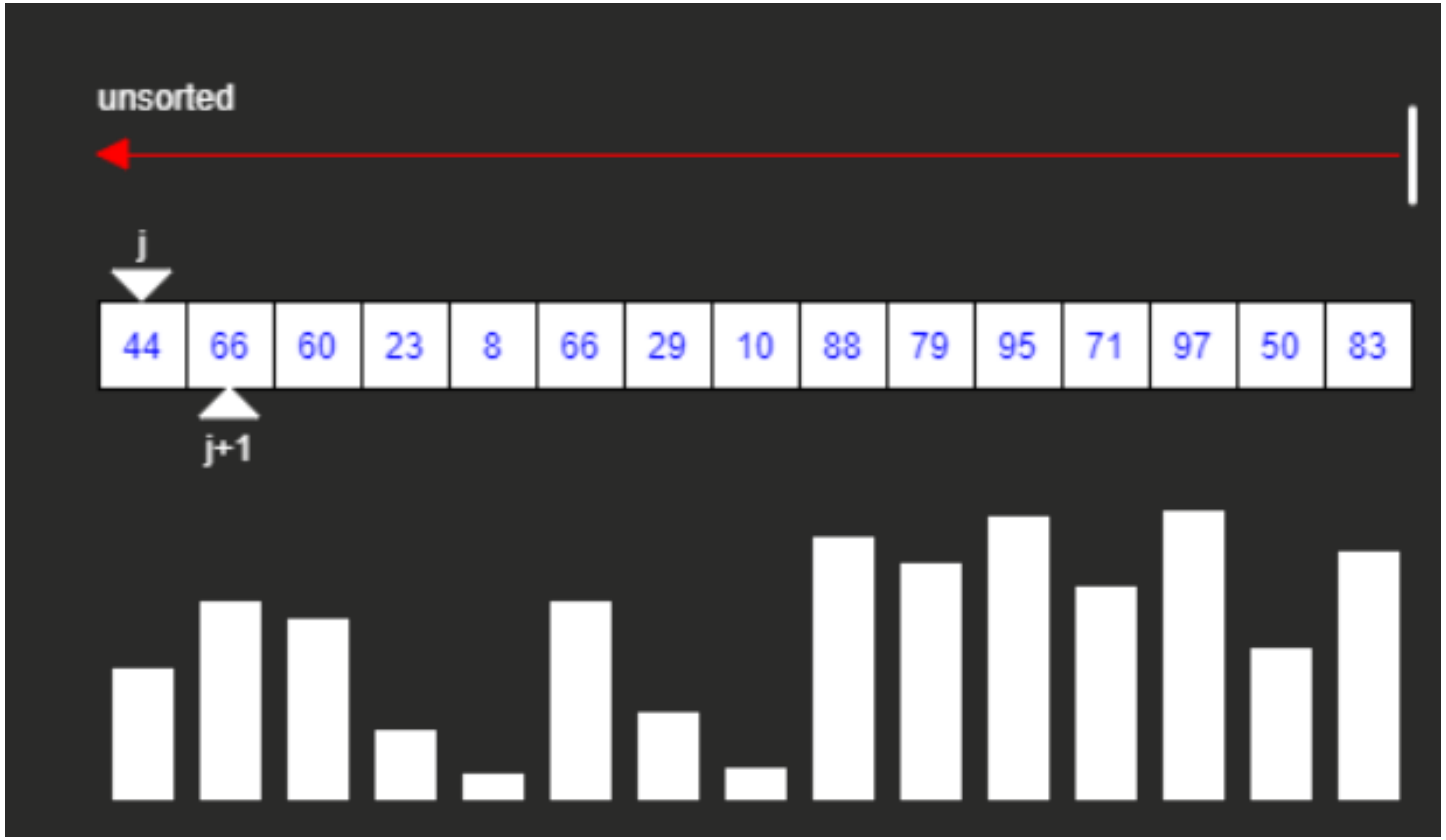
- A bubble sort is one of the simplest sorts to write.
- One of the slowest sorting algorithms available
- The idea behind a bubble sort is to start at the beginning of the array and swap adjacent elements that are not in order. Repeat this n-1 times where n is the size of the array and the array will be sorted
- There are several variations …

```
void bubble(int[] array)
{       int sz=array.Length;
        int i,j, tmp;

        for(i=0;i<sz-1;i++)
        {
                for(j=0;j<sz-i-1;j++)
                {
                        if(array[j] > array[j+1])
                        {
                                //swap arr[j] and arr[j+1]
                                tmp=array[j];
                                array[j]=array[j+1];
                                array[j+1]=tmp;
                        }
                }
        }
}
```

BUBBLE SORT – ANIMATION

- http://cathyatseneca.github.io/DSAnim/web/bubble.html



BUBBLE SORT – RUNNING TIME

- What is the running time of this algorithm?
- Can we improve this algorithm?
- What is the best/worst case scenario for this algorithm?
- What is the memory we need to carry out the operations of this algorithm?
  - O(1)

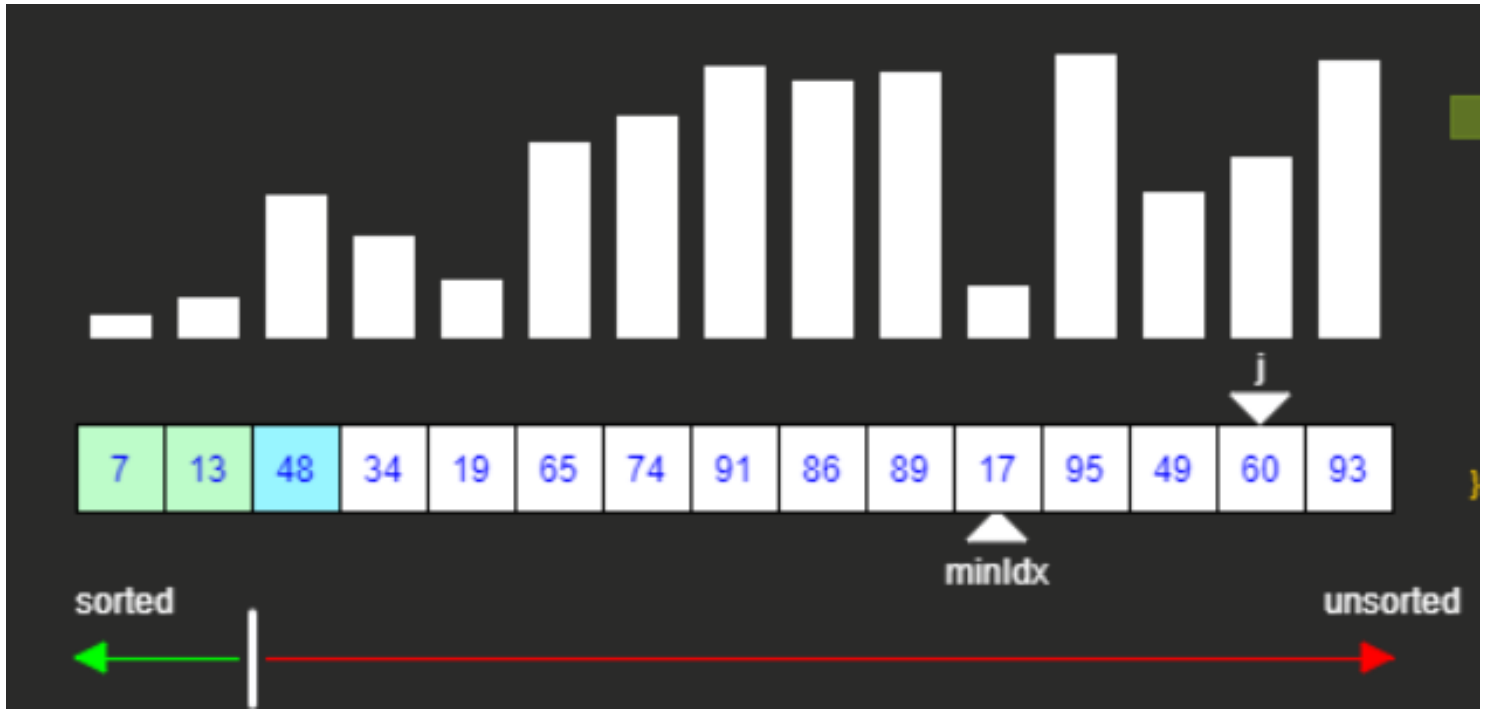
SELECTION SORT

- The selection sort algorithm works by selecting the smallest value in the unsorted portion of the array then swapping it with the first value of the unsorted portion of the array.
  - Starting at the beginning of the array, comparing the first element with the all other elements in the array.
  - The smallest element is placed in position 0, and the sort then begins again at position 1.
- There are several variations …

```
void selectionSort(int[] arr)
{
        int size=arr.Length;
        int minIdx, tmp;
        for(int i=0;i<size;i++)
        {
                minIdx=i;
                for(int j=i;j<size;j++)
                {
                        if(arr[j] < arr[minIdx])
                        {
                                minIdx=j;
                        }
                }
                //swap
                tmp=arr[i];
                arr[i]=arr[minIdx];
                arr[minIdx]=tmp;
        }
}
```

SELECTION SORT – ANIMATION

- http://cathyatseneca.github.io/DSAnim/web/selection.html



SELECTION SORT – RUNNING TIME

- What is the running time of this algorithm?
- Can we improve this algorithm?
- What is the best/worst case scenario for this algorithm?
- What is the memory we need to carry out the operations of this algorithm?
  - O(1)

INSERTION SORT

- The insertion sort algorithm works by looking at the array as if it is being filled in.
  - The idea is that the array starts off in the first iteration as if it has only one element in it.
  - The numbers that come after are then inserted into the "array" at the correct position.
  - This is continued until all values in the entire array have been properly "inserted".

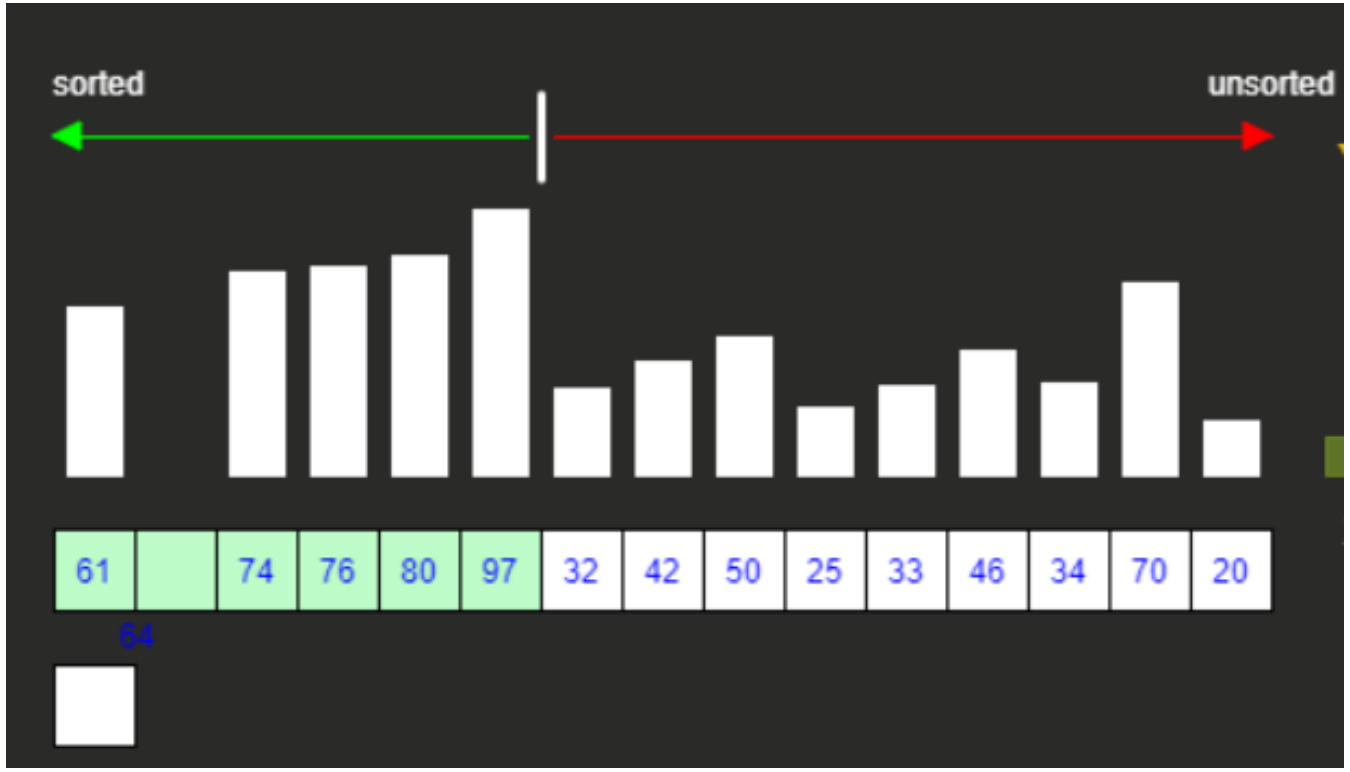
- Imagine that you have a stack of phone bills from the past two years and that you wish to organize them by date.
  - A fairly natural way to do this might be to look at the first two bills and put them in order.
  - Then take the third bill and put it into the right order with respect to the first two, and so on.
  - As you take each bill, you would add it to the sorted pile that you have already made..

- There are several variations …

```
void insertionSort(int[] arr)
{
        int size=arr.Length;
        int curr, i,j;
        for(i=0;i<size;i++)
        {
                curr=arr[i];
                for(j=i;j>0 && arr[j-1] > curr;j--)
                {
                        arr[j]=arr[j-1];
                }
                arr[j]=curr;
        }
}
```

INSERTION SORT – ANIMATION

- http://cathyatseneca.github.io/DSAnim/web/insertion.html



	i=1	2	3	4	5	6	7
42	20	17	13	13	13	13	13
20	42	20	17	17	14	14	14
17	17	42	20	20	17	17	15
13	13	13	42	28	20	20	17
28	28	28	28	42	28	23	20
14	14	14	14	14	42	28	23
23	23	23	23	23	23	42	28
15	15	15	15	15	15	15	42

INSERTION SORT – RUNNING TIME

- What is the running time of this algorithm?
- What is the best/worst case scenario for this algorithm?
- What is the memory we need to carry out the operations of this algorithm?
  - O(1)

COMPARING THESE SORTING ALGORITMS

- These three sorting algorithms are very similar in complexity and theoretically
  - At least, they should perform similarly when compared with each other.
  - We can use the Timing class to compare these three algorithms
  - Experimental results: …

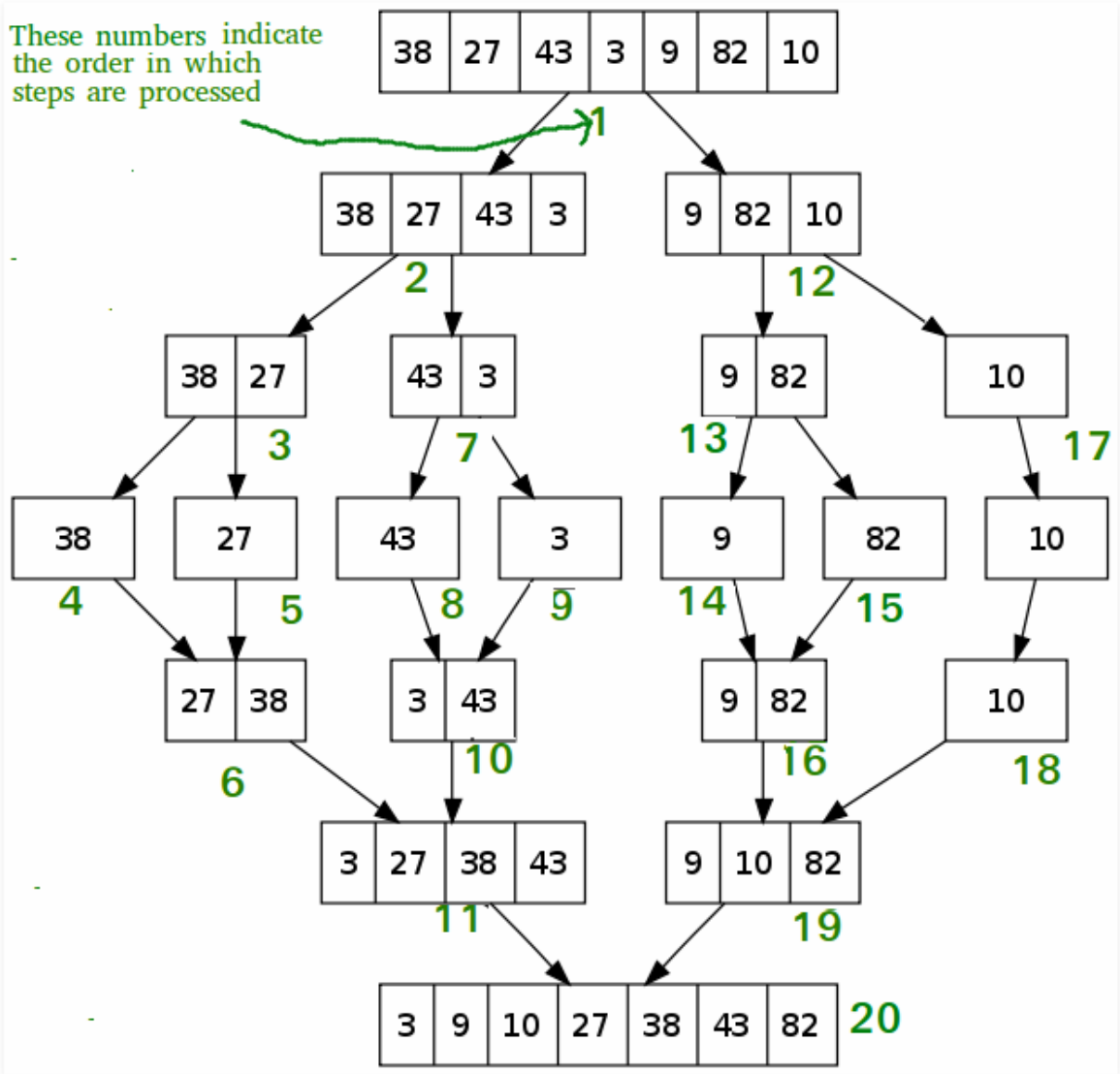
	Insertion	Bubble	Selection
Comparisons:			
Best Case	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
Average Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Worst Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$

MERGE SORT

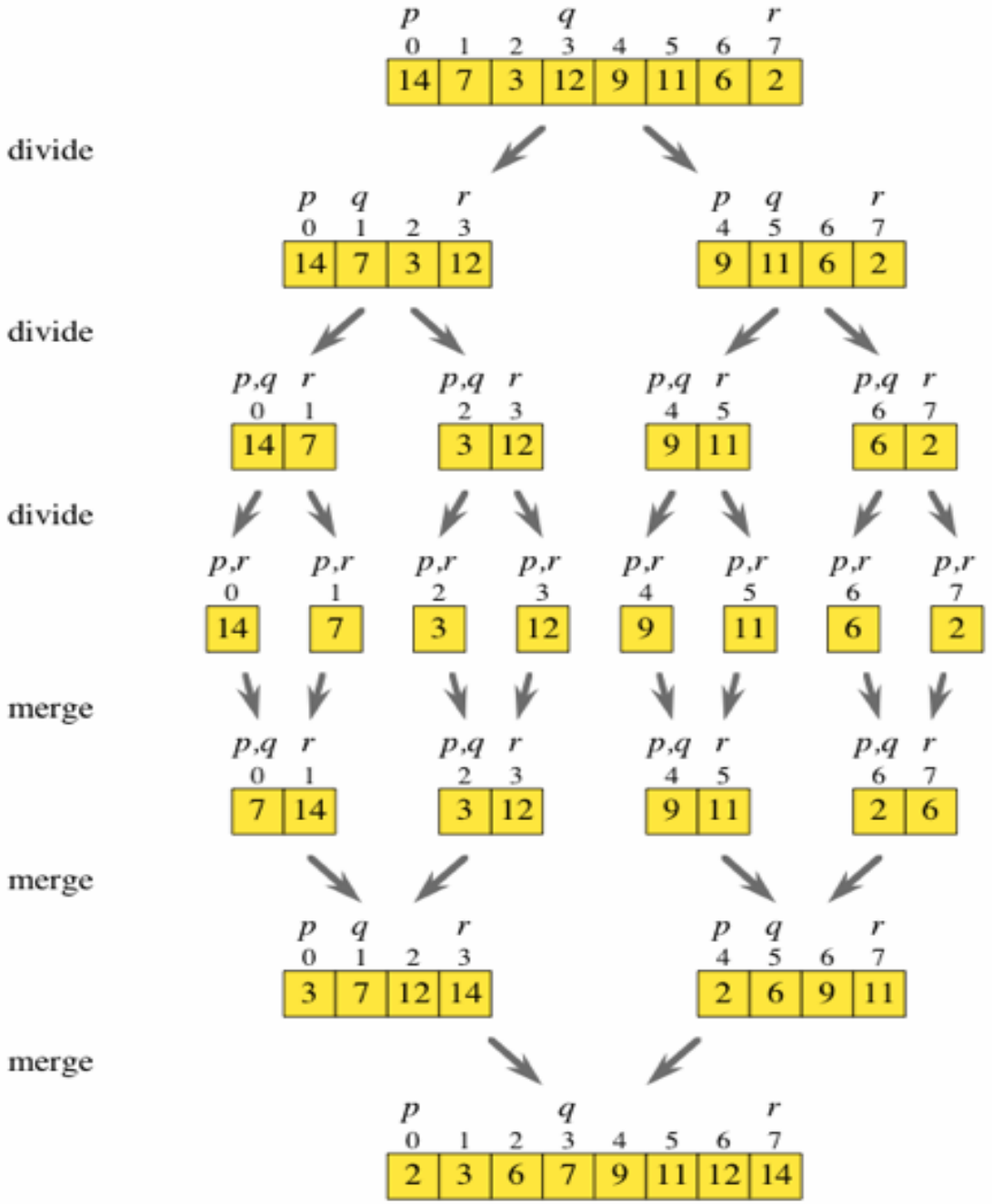
- A natural approach to problem solving is **divide and conquer.**
  - The idea behind merge sort: split the list in half, sort the halves, and then merge the sorted halves together.
  - Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted.
  - An example of a recursive algorithm.
    - Can also be written iteratively

- **divide-and-conquer** paradigm:
  - Divide the problem into a number of subproblems.
  - Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
  - Combine the solutions to the subproblems into the solution for the original problem.

- There are several variations …



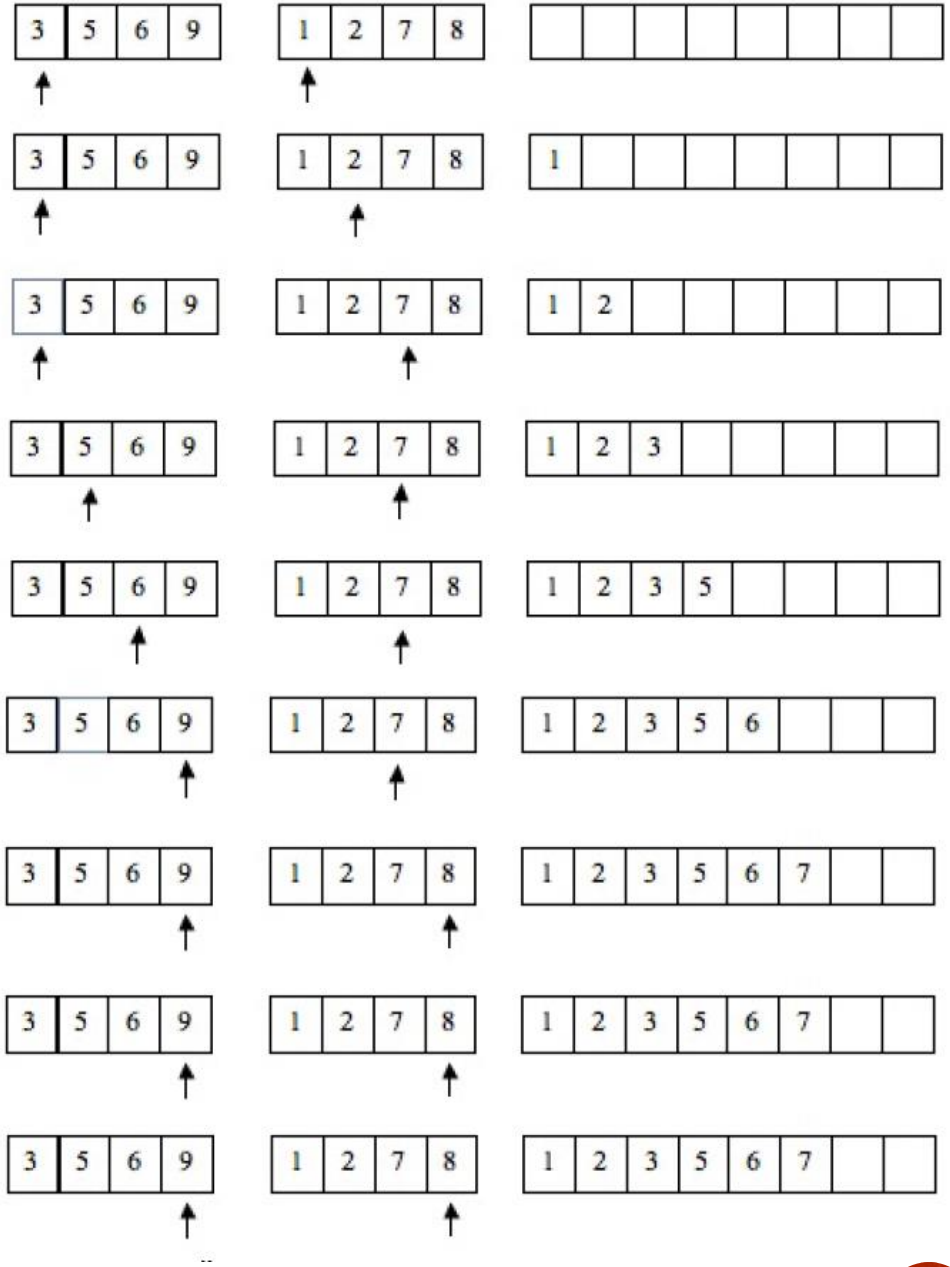
MERGE SORT EXAMPLE



let's choose 7 unsorted numbers and do a merge sort on them

MERGE ALGORITHM

- Have a way to "point" at the first element of each of the two list
- compare the values being pointed at and pick the smaller of the two
- copy the smaller to the merged list, and advance the "pointer" of just that list to the next item.
- Continue until one of the list is completely copied then copy over remainder of the rest of the list



MERGE SORT ALGORITHM (IN C#)

```
static void MergeSort(int arr[])
{
        int[] tmp=new int[arr.Length];
        MSort(arr,tmp,0, arr.Length -1);
}
```

```
static void MSort(int[] arr,int[] tmp, int start,int end)
{
        if(start<end)
        {
                int mid=(start+end)/2;
                MSort(arr,tmp,start,mid);
                MSort(arr,tmp,mid+1,end);
                Merge(arr,tmp,start,mid+1,end);
        }
}
```

```
static void Merge(int[] arr, int[] tmp, int startA, int startB, int endB)
{
        int aptr=startA;
        int bptr=startB;
        int idx=startA;

        while(aptr < startB && bptr < endB+1)
        {
                if(arr[aptr] < arr[bptr])
                        tmp[idx++]=arr[aptr++];
                else
                        tmp[idx++]=arr[bptr++];
        }

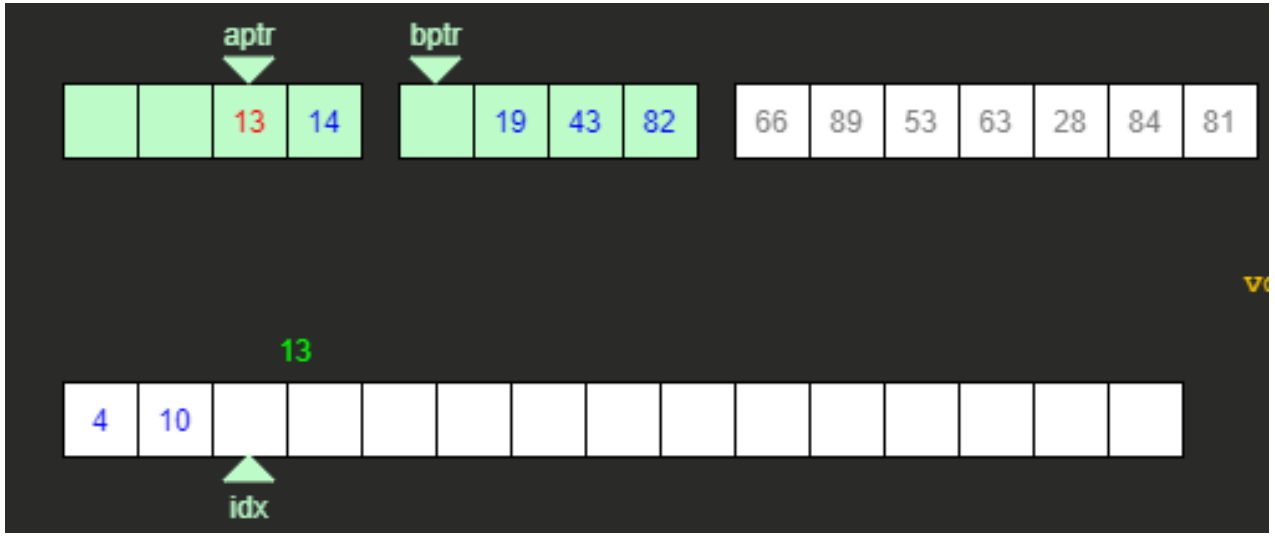
        while(aptr<startB)
                tmp[idx++]=arr[aptr++];

        while(bptr < endB+1)
                tmp[idx++]=arr[bptr++];

        for(int i=startA;i<=endB;i++)
                arr[i]=tmp[i];
}
```

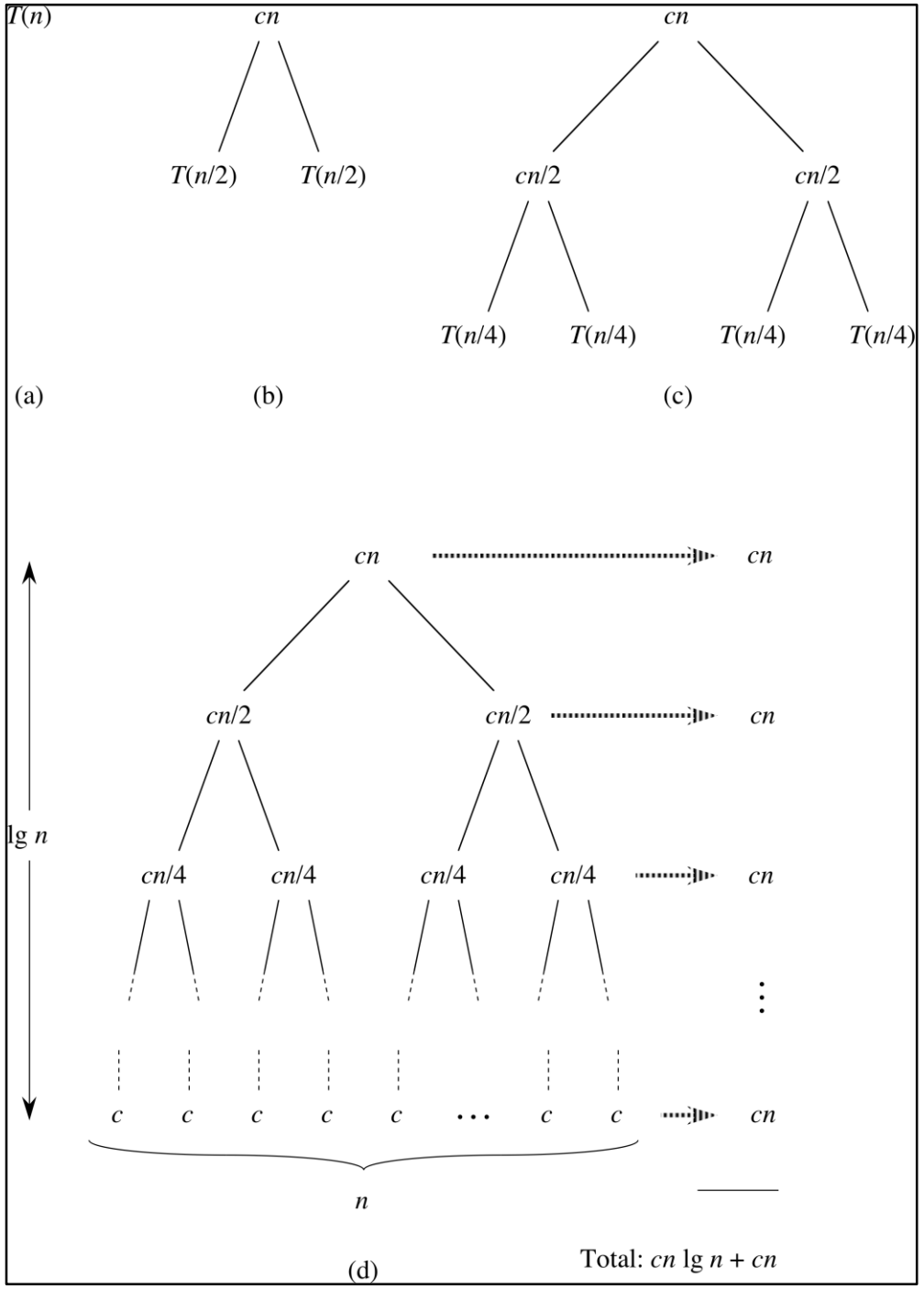
MERGE SORT - ANIMATION

- http://cathyatseneca.github.io/DSAnim/web/merge.html



MERGE SORT RUNNING TIME

- What is the running time of this algorithm?
- What is the best/worst case scenario for this algorithm?
- What is the memory we need to carry out the operations of this algorithm?



QUICK SORT

- This sort is fast and does not have the extra memory requirements of MergeSort.
  - On average its run time is O(n log n)
    - When subarrays are balanced
  - It does have a worst case run time of $O(n^2)$
    - When the subarrays are unbalanced

- Quicksort is aptly named because, when properly implemented, it is the fastest known general-purpose in-memory sorting algorithm in the average case.
  - It does not require the extra array needed by Mergesort, so it is space efficient as well.
    - Sorts in place

- There are several variations … various ways of selecting a pivot … (first, last, middle,…)

QUICK SORT – IDEA

- You have to alphabetize a stack of student papers.

- You will pick a letter that is in the middle of the alphabet, such as M (called pivot), putting (partitioning) student papers whose name starts with A through M in one stack and names starting with N through Z in another stack.
- Then you (recursively) split the A-M stack into two stacks and the N-Z stack into two stacks using the same technique.
  - Then you do the same thing again until you have a set of small stacks (A–C, D–F, . . . , X–Z) of two or three elements that sort easily.

- Once the small stacks are sorted, you simply put all the stacks together and you have a set of sorted papers.
  - This process is recursive
    - Each stack is broken up into smaller and smaller stacks.
    - Once a stack is broken down into one element, that stack cannot be further broken up and the recursion stops.

QUICK SORT – ALGORITHM

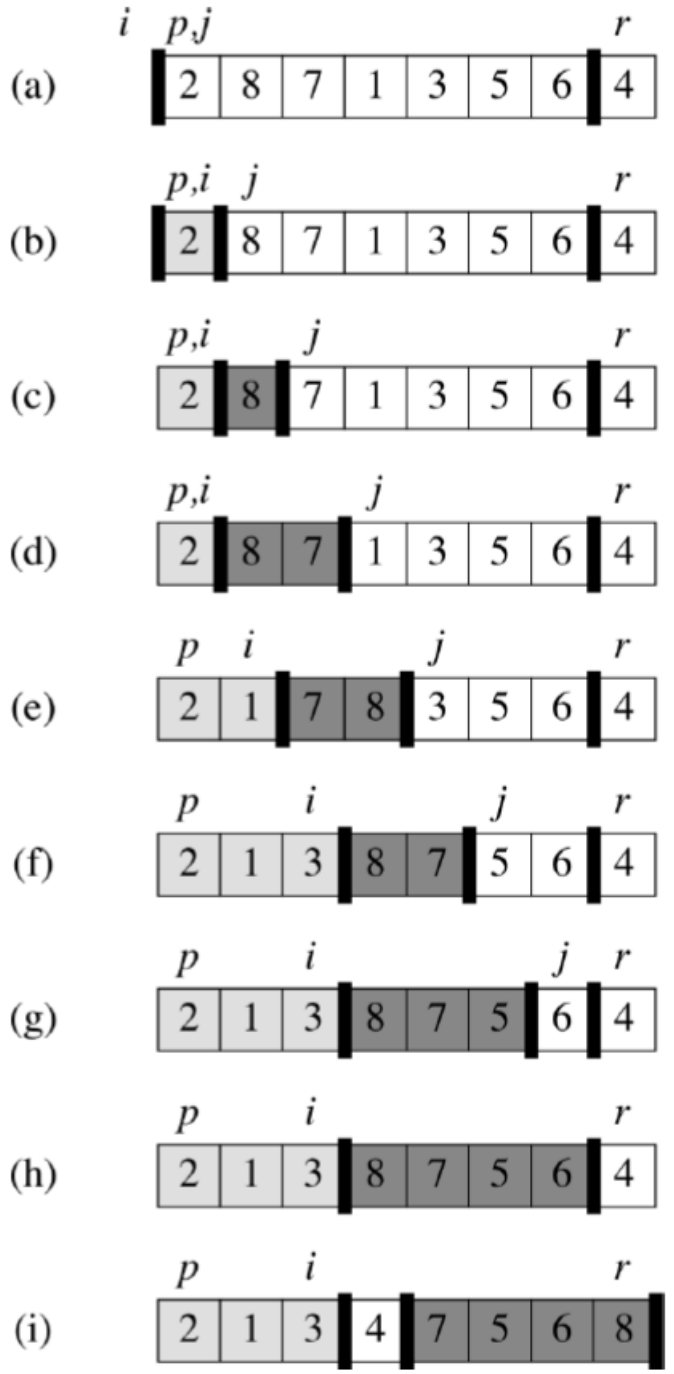
- It is based on the three-step process of divide-and-conquer. To sort array A[p...r]:
  - **Divide**: Partition A[p ... r], into two (possibly empty) subarrays A[p . . q −1] and A[q + 1 . . r], such that:
    - each element in the first subarray A[p ... q −1] is ≤ A[q] and
    - A[q] is ≤ each element in the second subarray A[q + 1 ... r].
  - **Conquer**: Sort the two subarrays by recursive calls to QUICKSORT.
  - **Combine**: No work is needed to combine the subarrays, because they are sorted in place.

- **QUICKSORT(A, p, r)**
  if p < r
  q ←PARTITION(A, p, r)
  QUICKSORT(A, p, q −1)
  QUICKSORT(A, q + 1, r)
- Initial call is **QUICKSORT(A, 0, n-1)**.
- **PARTITION** selects the last element **A[r]** in the subarray A[p . . r] as the pivot
  - Alternatively, it could pick **A[p]** or **A[(p+r)/2]**.

QUICK SORT – PARTITION

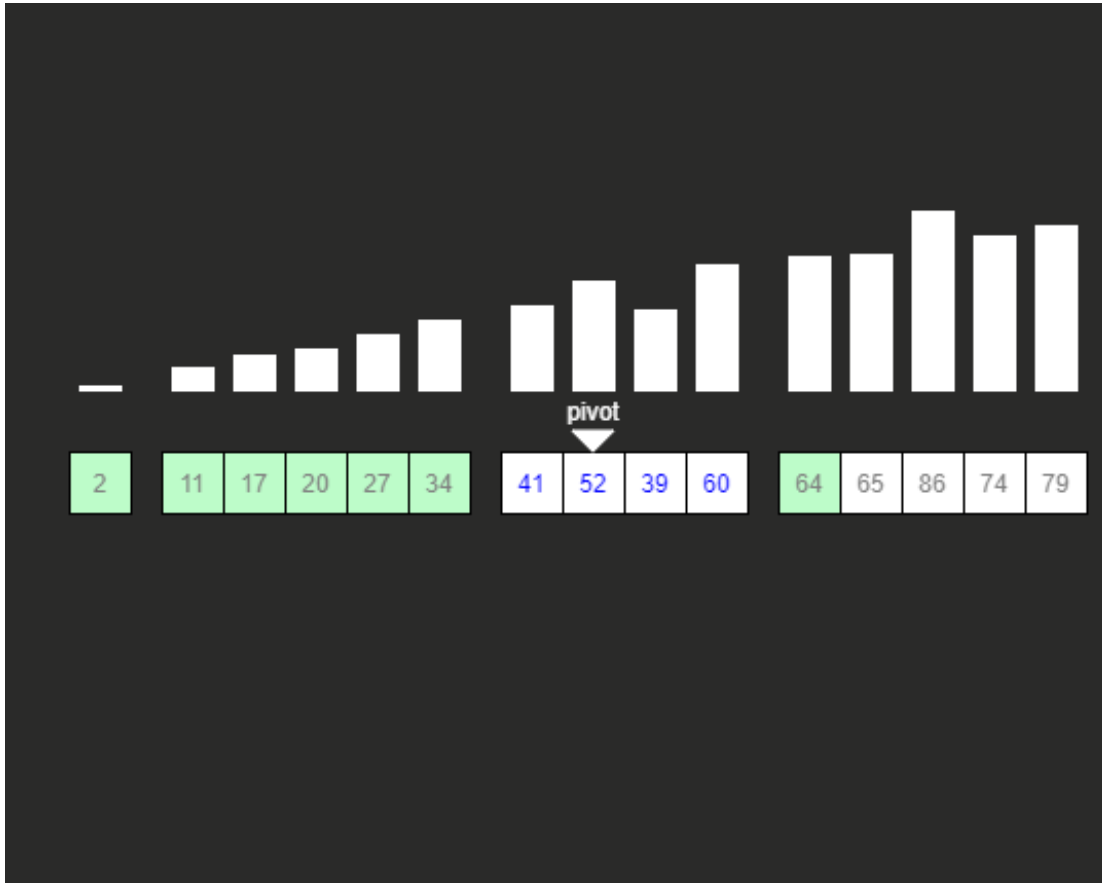
- **PARTITION(*A, p, r*)**

```
x ←A[r]                          // x = pivot
i←p −1                           // i = pos of last element< x
for j ←p to r −1                 // j = current element
        if A[j] ≤ x
                i←i+1
                exchange A[i] ↔ A[j ]
exchange A[i+1] ↔A[r]
return i+1                       //return new pos of pivot
```



QUICK SORT – ANIMATION

- http://cathyatseneca.github.io/DSAnim/web/quick.html




```
void quick(int arr[],int left,int right){
  if(left<right){
    int sz=right-left+1;
    int pivotpt=(left+right)/2;
    int i=left;
    int j=right-1;
    int pivot=arr[pivotpt];
    swap(arr[pivotpt],arr[right]);
    pivotpt=right;
    while(i<j){
      while(i< right-1 && arr[i]<pivot)
        i++;
      while(j > 0 && arr[j]>pivot)
        j--;
      if(i<j)
        swap(arr[i++],arr[j--]);
    }
    if(i==j && arr[i] < arr[pivotpt])
      i++;
    swap(arr[i],arr[pivotpt]);
    quick(arr,left,i-1);
    quick(arr,i+1,right);
  }
}
void quick(int arr[],int size){
  quick(arr,0,size-1);
}
```

QUICK SORT – RUNNING TIME

- What is the running time of this algorithm?
- What is the best/worst case scenario for this algorithm?
  - Does it matter how we pick the pivot?
- What is the memory we need to carry out the operations of this algorithm?
  - In-memory

OTHER SORTING ALGORITHMS

- Can we sort in O(n)?
- Radix sort, shell sort, Hashing
- Heap sort, tree sort – to come …

USING THE C# LIBRARY

- How can we sort an array using the library?
  - Try:

```
int[] arr = { 1,6,3,5,7};
Array.Sort(arr);
```

- How can we search an array using the library?
  - Try:

```
private static bool match(int x)
{        return x == 6;       }
```

  - Then you can use it (as a delegate) like below:
    - int[] arr1 = { 10, 2, 6, 3, 7, 17, 13, 3, 66 };
    - Console.WriteLine(Array.FindIndex(arr1, match));
  - For more info look into:
    - https://docs.microsoft.com/en-us/dotnet/api/system.array.findindex?view=netframework-4.7.2

THE MASTER METHOD

- **SIMPLIFICATION!!!**
  Let a ≥ 1 and b > 1 be constants, let f (n) be a function, and let T (n) be defined on the nonnegative integers by the recurrence

$$\boldsymbol{T(n) = aT(n/b) + f(n)},$$

  where we interpret n/b to mean either ⌊n/b⌋ or ⌈n/b⌉. Let $f(n) = \Theta(n^c)$

- If $c < \log_b a$ (using big-Oh notation) then $T(n) = \Theta\left(n^{\log_b a}\right)$
- If $c = \log_b a$ then $T(n) = \Theta\left(n^{\log_b a} \log n\right)$
- If $c > \log_b a$ then $T(n) = \Theta(n^c)$ ("regularity" condition skipped)

- Note: we compare the function f(n) with $n^{\log_b a}$.
  The larger of the two determines the solution. If equal, we multiply by a logarithmic function

INTERVIEW QUESTIONS (JUST FOR PRACTICE!)

- What is a linear search?

- a sorted array with all elements appear twice except one, find that element (the best solution should be O(logn)) [Glassdoor.com: Microsoft interview question] 10 hw extra points available

- Find out whether a substring occurs in a given string?
  [Glassdoor.com: Microsoft interview question] 10 hw extra points available

- If you had 100 floors and 2 eggs, what is the most efficient way to determine what floor is the maximum height you can drop an egg and not crack it?
  [Glassdoor.com: Microsoft interview question]

INTERVIEW PROBLEMS (JUST FOR PRACTICE!)

- Return sum of the largest two numbers in a given array.
  10 homework extra points available

INTERVIEW QUESTIONS (JUST FOR PRACTICE!)

- How does a selection sort work for an array?
- Which sorting algorithm is considered the fastest?
- What is a bubble sort and how do you perform it?
- How does selection sort work?

INTERVIEW PROBLEMS (JUST FOR PRACTICE!)

- Write a Program which checks if two Strings are Anagram or not?
  5 homework extra points available

- Given a number (e.g.: 427) find the largest possible number from its digits (e.g.: 742)
  5 homework extra points available
- find repeated elements in an array [Glassdoor.com: Microsoft interview question]
  5 homework extra points available
- shuffle a string array [Glassdoor.com: Microsoft interview question]
  5 homework extra points available
- sort 2 different arrays, and save the result in the second array, if the first contains n elements, and the second contains m elements, and the second is allocated a length of m+n. [Glassdoor.com: Microsoft interview question]
  10 homework extra points available – do not use an extra array … the second array should be sufficient i.e. your algorithm should use O(1) extra space .

PROGRAMMING COMPETITION

- CCSC SE 2011:
  10 homework extra points available

Problem 1

SMILE! :-)

September 19, 1982 — On this day, a Carnegie Mellon University computer science professor named Scott E. Fahlman posted a smiley face **:-)** on an electronic message board.

Not just content to remain a frozen smirk, it has learned to wink **;-)** and stick out its tongue **:-p**. Sometimes it gets sad **:-(** or confused **:-S**, and sometimes it's scared **=:-0** but most of the time it's cheerful **:-D**.

As it grew older, it had its first kiss **:-*** and smoked its first cigarette **:-Q** and even grew its hair out **&:-)**. It's shown its fashion sense, too, sporting sunglasses **B-)** and baseball caps **d:-)**, and even the occasional bow tie **:-)8**.

Your job is to count the number of original smiley faces in a single line of input. That is, the number of times the substring "**:-)**" occurs.

Input

Input consists of a single string of length n, ($2 \leq n \leq 80$). Assume the last character in the file will always be an end of line character.

Output

Output the total times the substring "`:-)`" occurs on a given line of input. The output should be formatted exactly as shown below.

Sample Input

```
From: <Fahlman@Cmu>:-D :-) Read it sideways::-)-) ;-)Happy:-)8 Coding!
```

Output Corresponding to Sample Input

```
Smiley Count = 3
```

PROGRAMMING COMPETITION

- CCSC SE 2011:
  15 homework extra points available

Problem 2

Look and Say

The look and say sequence is defined as follows. Start with any string of decimal digits as the first element in the sequence. Each subsequent element is defined from the previous one by "verbally" describing the previous element. For example, the string 122344111 can be described as "one 1, two 2's, one 3, two 4's, three 1's". Therefore, the element that comes after 122344111 in the sequence is 1122132431. Similarly, the string 101 comes after 1111111111.

Notice that it is generally not possible to uniquely identify the previous element of a particular element. For example, a string of 112213243 1's also yields 1122132431 as the next element.

Input & Output

The first line of input will be a number n, ($2 \leq n \leq 20$), telling how many cases are to follow. The next n-lines consist of a number of cases. Each case consists of a line of up to 1000 digits. For each test case, print the string that follows the given string. Assume all input will consist of decimal digits 0 – 9.

Sample Input

```
3
122344111
1111111111
12345
```

Output Corresponding to Sample Input

```
1122132431
101
1112131415
```

PROGRAMMING COMPETITION

- CCSC SE 2012:
  10 homework extra points available

Problem 1
Nth Largest Value

For this problem, you will write a program that prints the *Nth* largest value in a fixed sized array of integers. To make things simple, *N* will be 3 and the array will always have 10 decimal integer values.

Input
The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of test cases that follow. Each test case consists of a single line containing the test case number, followed by a space, followed by ten decimal integers whose values are between 1 and 1000 inclusive. Each of the ten integers will always be separated by a single space.

Output
For each test case, generate one line of output with the following values: the test case number, a space, and the 3^{rd} largest value of the corresponding 10 integers.

Sample Input
```
4
1 1 2 3 4 5 6 7 8 9 1000
2 338 304 619 95 343 496 489 116 98 127
3 931 240 986 894 826 640 965 833 136 138
4 940 955 364 188 133 254 501 122 768 408
```

Output Corresponding to Sample Input
```
1 8
2 489
3 931
4 768
```

PROGRAMMING COMPETITION

- CCSC SE 2014:
  10 homework extra points available

Problem 1

Royal Tweet

On October 24th, 2014, Britain's Queen Elizabeth II dipped a toe into 21st-century communications when she posted her first tweet. Signing herself Elizabeth R., for regina or queen, she welcomed visitors to a new Information Age gallery focused on the evolution of modern communications at the Science Museum in London. The inaugural tweet (shown above) was posted to the official British monarchy Twitter feed, which has more than 700,000 followers.

Your job is to write a program to scan a given number of tweets for how many mention the at symbol to the British Monarchy, `@BritishMonarchy`. The username following the `@` symbol in Twitter is case insensitive.

Input
The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of tweets that follow. Each tweet consists of a single line of input containing a string. Each string is of length n, ($1 \leq n \leq 140$). All tweets are separated by a single blank line.

Output
Output the total tweets that contain the substring `@BritishMonarchy` (regardless of case) exactly in the format below. A given tweet that mentions `@BritishMonarchy` more than once is legal and should not be double counted.

Sample Input

```
4
So excited that we can now tweet The Queen @bRitishMonaRchY!! :-)

@BritishMonarchy #Christmas is 6 weeks away!! #soClose

Russ Rose would like to wish you all a Happy #Thanksgiving. #SNL

@BritishMonarchy's great grandson, heir to throne of @BritishMonarchy
```

Output Corresponding to Sample Input

```
Total Tweets Containing @BritishMonarchy = 3
```

PROGRAMMING COMPETITION

- CCSC SE 2015:
  15 homework extra points available

Problem 1

Letters Not Used

For this problem, you are interested in displaying a list of all letters that do not appear in a given input string. The letters in the output will be lowercase, regardless of the case of the letters in the input string.

Input

The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of test cases that follow. Each test case consists of a single line of input containing a string. Each string is of length n, ($1 \leq n \leq 80$). All input strings will consist solely of letters and spaces, no punctuation or special symbols.

Output

For each case, you should print out the statement :

```
Letters missing in case x:
```

where x is the case number. Place a colon immediately after the case number along with a single space. This is followed by all the letters which are not used in the input. All letters should be lowercase and in ascending sorted order from a to z.

Sample Input

```
5
The quick brown fox jumped over the lazy dog
How much better wood would a woodchuck chuck today than yesterday
A skunk sat on a stump and thunk the stump stunk
Pack my box with five dozen liquor jugs
Razorback jumping frogs can level six piqued gymnasts
```

Output Corresponding to Sample Input

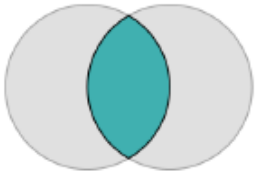
```
Letters missing in case #1: s
Letters missing in case #2: fgijpqvxz
Letters missing in case #3: bcfgijlqrvwxyz
Letters missing in case #4:
Letters missing in case #5: hw
```

PROGRAMMING COMPETITION

- CCSC SE 2017:
  15 homework extra points available

Problem 8

Overlap Length



Write a program that compares two strings, *string1* and *string2*, and calculates their overlap length. For example, the strings "FURMAN" and "MANKIND" have an overlap length of 3. Overlap length is the string of the longest length at the end of *string1* which can also be found starting at the beginning of *string2*. "APPLE" and "LEMON" have an overlap length of 2, while "LEMON" and "APPLE" have an overlap length of 0. "ALE" and "APPLE" would have an overlap length of 0, while "ALE" and "ALE" have an overlap length of 3.

Input
The input will start with a single line containing a positive integer *n* representing the specified number of pairs of lines to be input. Each pair of lines consists of two strings starting with a single character that is not a white space character (tab, blank, newline). This single character is always a letter, and will be followed by zero or more letters. The last character on each line is a newline character. All strings should be converted to uppercase upon input, and before comparison. Each string will consist of 1 to 80 letters.

Output
Format output as shown below where *string1* and *string2* both appear in uppercase with the string " `and` " between them, and the string " `have an overlap length of` " following. Note that *string1* is always the first string of the two strings to be input.

Sample Input

```
5
football
ball
ALE
Apple
Clemson
son
SC
C
APPLE
LEMON
```

Output Corresponding to Sample Input

```
FOOTBALL and BALL have an overlap length of 4
ALE and APPLE have an overlap length of 0
CLEMSON and SON have an overlap length of 3
SC and C have an overlap length of 1
APPLE and LEMON have an overlap length of 2
```

PROGRAMMING COMPETITION

- CCSC SE 2012:

Problem 1

Nth Largest Value

For this problem, you will write a program that prints the *Nth* largest value in a fixed sized array of integers. To make things simple, *N* will be 3 and the array will always have 10 decimal integer values.

Input

The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of test cases that follow. Each test case consists of a single line containing the test case number, followed by a space, followed by ten decimal integers whose values are between 1 and 1000 inclusive. Each of the ten integers will always be separated by a single space.

Output

For each test case, generate one line of output with the following values: the test case number, a space, and the 3^{rd} largest value of the corresponding 10 integers.

Sample Input

```
4
1 1 2 3 4 5 6 7 8 9 1000
2 338 304 619 95 343 496 489 116 98 127
3 931 240 986 894 826 640 965 833 136 138
4 940 955 364 188 133 254 501 122 768 408
```

Output Corresponding to Sample Input

```
1 8
2 489
3 931
4 768
```

PROGRAMMING COMPETITION

- CCSC SE 2013:
  10 homework extra points available

Problem 1

Ordered Lists

For this problem, you will write a program that scans a list of integers to verify whether or not it is ordered. To make things simple, each of the lists will always have 10 integer values. A list is said to be in *ascending order* if all 10 values are listed in order from least to greatest. A list is said to be in *descending order* if all 10 values are listed in order from greatest to least. For this problem, there must be at least two different values in the list for it to be ordered. That is, a list with all duplicate values is not ordered.

Input

The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of test cases that follow. Each test case consists of a single line containing exactly ten integers. Each of the ten integers will always be separated by a single space.

Output

For each test case, generate one line of output in the exact format below indicating the list number, and whether the list is in ascending order, descending order, or not ordered. Do not forget the ending period.

Sample Input

```
5
1 2 3 4 5 6 7 8 9 10
50 39 28 15 9 5 3 2 1 0
5 5 5 5 5 5 5 5 5 5
5 6 7 5 4 2 5 1 5 0
5 6 7 8 9 -1 -2 -3 -4 -5
```

Output Corresponding to Sample Input

```
List 1 is in ascending order.
List 2 is in descending order.
List 3 is not ordered.
List 4 is not ordered.
List 5 is not ordered.
```

PROGRAMMING COMPETITION

- CCSC SE 2015:
  10 homework extra points available

Problem 8

Diverse Matrix?

A two-dimensional array is *diverse* if no two of its rows have entries that sum to the same value. In the following examples, the first array is diverse because each row sum is different, but the second array is not diverse because the first and last rows have the same sum.

1	3	2	7	3
10	10	4	6	2
5	3	5	9	6
7	6	4	2	1

1	1	5	3	4
12	7	6	1	9
8	11	10	2	5
3	2	3	0	6

Input

The first line of input contains a single integer n, ($1 \leq n \leq 1000$), which is the number of test matrices that follow. For each matrix being tested, the input will be a single line containing two integers separated by a space representing the number of rows r and columns c in the range between 1 and 10 inclusive, ($1 \leq r, c \leq 10$). This is followed by r rows each of c integers each separated by a space. All input is in row major order. Each integer is a valid 32-bit integer.

Output

For each matrix, output a single word `yes` or `no` in lowercase indicating whether it is diverse.

Sample Input

```
2
4 5
1 3 2 7 3
10 10 4 6 2
5 3 5 9 6
7 6 4 2 1
4 5
1 1 5 3 4
12 7 6 1 9
8 11 10 2 5
3 2 3 0 6
```

Output Corresponding to Sample Input

```
yes
no
```

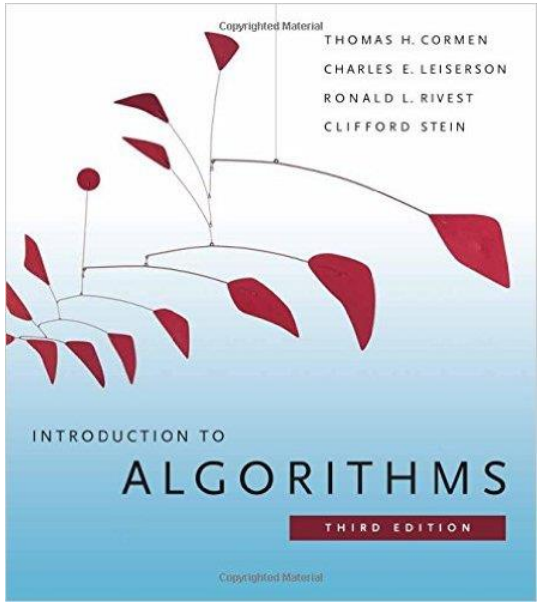
SOLVING RECURRENCES – NOT FOR TEST

There are three main methods for solving recurrences (i.e. for obtaining asymptotic bounds)

- **Substitution method**: we guess a bound and then prove it using mathematical induction

- **Recursion-Tree method**: uses a tree whose nodes represent costs incurred at various levels of the recursion. Then we can use various techniques for bounding summations to solve the recurrence.

- **The Master Method** (**very important!!!**): provides bounds for recurrences of the form
  $T(n) = aT(n/b) + f(n)$ where $a \geq 1, b > 1$.
  - The divide and conquer algorithm creates a subproblems, each of which is $1/b$ the size of the original problem, and in which the divide and combine steps together take $f(n)$ time.

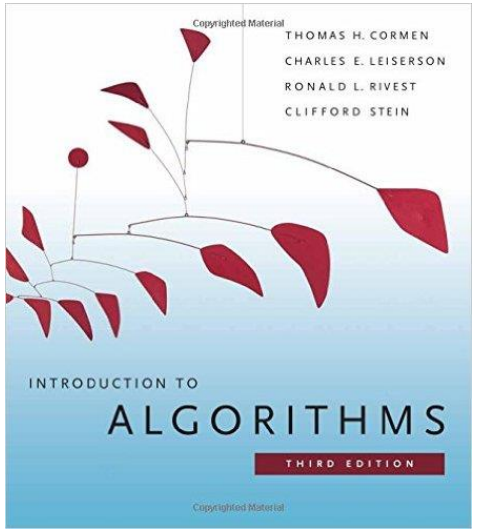


SUBSTITUTION METHOD – NOT FOR TEST

Two steps:

- Guess the form of the solution
- Use mathematical induction to find the constants and show that the solution works

Powerful method, but we must be able to guess the form of the answer in order to apply it.



SUBSTITUTION METHOD – NOT FOR TEST

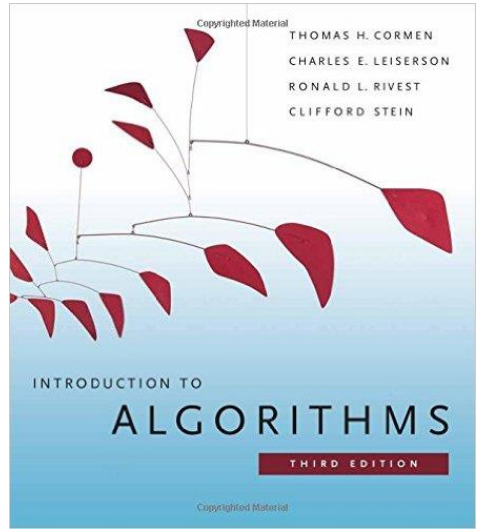
Find the upper bound for the recurrence: $T(n) = 2\,T(\lfloor n/2 \rfloor) + n$

We guess that $T(n) = O(n \log n)$

- how did we get this guess?
- there is no general way to guess the correct solutions to recurrences.
- guessing a solution takes experience and, occasionally, creativity
- can also use recursion trees, which we'll see soon

To prove that $T(n) = O(n \log n)$ we need to show that $T(n) \leq c \cdot n \cdot \log n$ for some c and all $n \geq (some)\ n_0$.

- **Note1**: we need to find c and n_0
- **Note2**: out proof will look slightly different than what's in the book.



SUBSTITUTION METHOD – NOT FOR TEST

We want to show that $T(n) \leq c \cdot n \cdot \log n$ for some c and all $n \geq (some)\ n_0$.

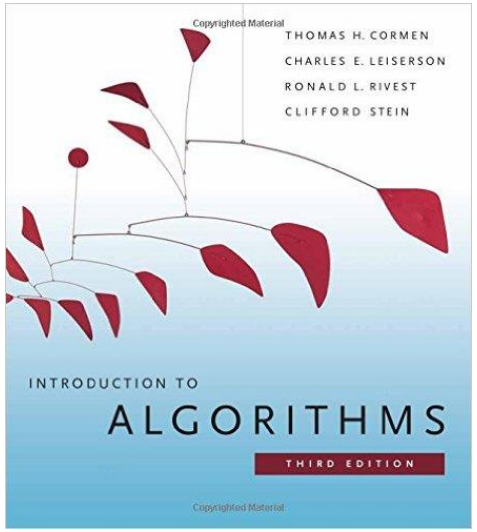
Assuming this is true, we'll have that $T(\lfloor n/2 \rfloor) \leq c \cdot \lfloor n/2 \rfloor \cdot \log \lfloor n/2 \rfloor$ hence

$$
\begin{aligned}
T(n) &= 2\,T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\
&\leq 2c \left\lfloor \frac{n}{2} \right\rfloor \log \left\lfloor \frac{n}{2} \right\rfloor + n \\
&= cn \log \left\lfloor \frac{n}{2} \right\rfloor + n \\
&= cn \log n - cn \log 2 + n \\
&\leq cn \log n,
\end{aligned}
$$

for all $n \geq 1$ and $c \geq \frac{1}{log2} = 1$.

Note: we are not given any value for T(1). If we assume T(1)=1, then we get $T(1) \leq c \cdot 1 \cdot \log 1 = 0$ which is a contradiction. In these cases we can let n be larger than 1 to avoid this contradiction. We also choose c large enough so that the base cases of n=2 and n=3 work.

Proof by induction: let T(1)=??? Then T(2)=????, T(3)=??? Once base case works we do the induction step…



SUBSTITUTION METHOD – NOT FOR TEST

We want to show that $T(n) \leq c \cdot n \cdot \log n$ for some c and all $n \geq (some)\ n_0$.

Assuming this is true, we'll have that $T(\lfloor n/2 \rfloor) \leq c \cdot \lfloor n/2 \rfloor \cdot \log \lfloor n/2 \rfloor$ hence

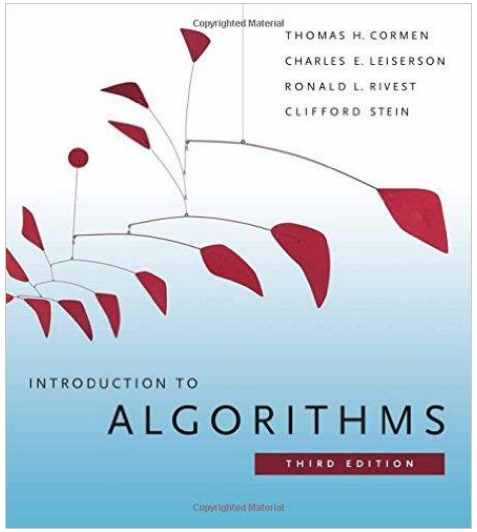
$$\begin{aligned} T(n) &= 2\,T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\ &\leq 2c \left\lfloor \frac{n}{2} \right\rfloor \log \left\lfloor \frac{n}{2} \right\rfloor + n \\ &= cn \log \left\lfloor \frac{n}{2} \right\rfloor + n \\ &= cn \log n - cn \log 2 + n \\ &\leq cn \log n, \end{aligned}$$

for all $n \geq 1$ and $c \geq \frac{1}{log2} = 1$.

Note: we are not given any value for T(1). If we assume T(1)=1, then we get $T(1) \leq c \cdot 1 \cdot \log 1 = 0$ which is a contradiction. In these cases we can let n be larger than 1 to avoid this contradiction. We also choose c large enough so that the base cases of n=2 and n=3 work.

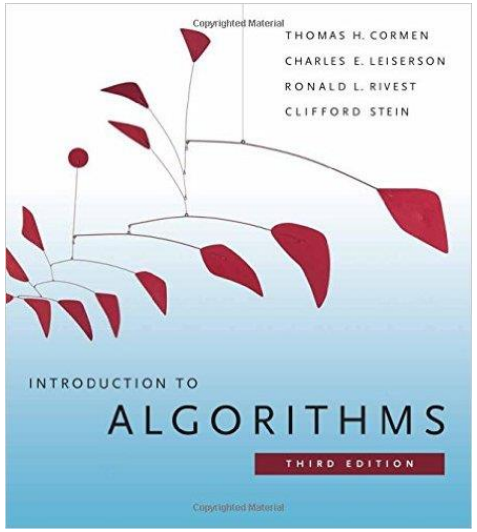
Proof by induction: let T(1)=??? Then T(2)=????, T(3)=??? Once base case works we do the induction step…

Example 2: Show that the solution of $T(n) = T(n-1) + n$ is $O(n^2)$.

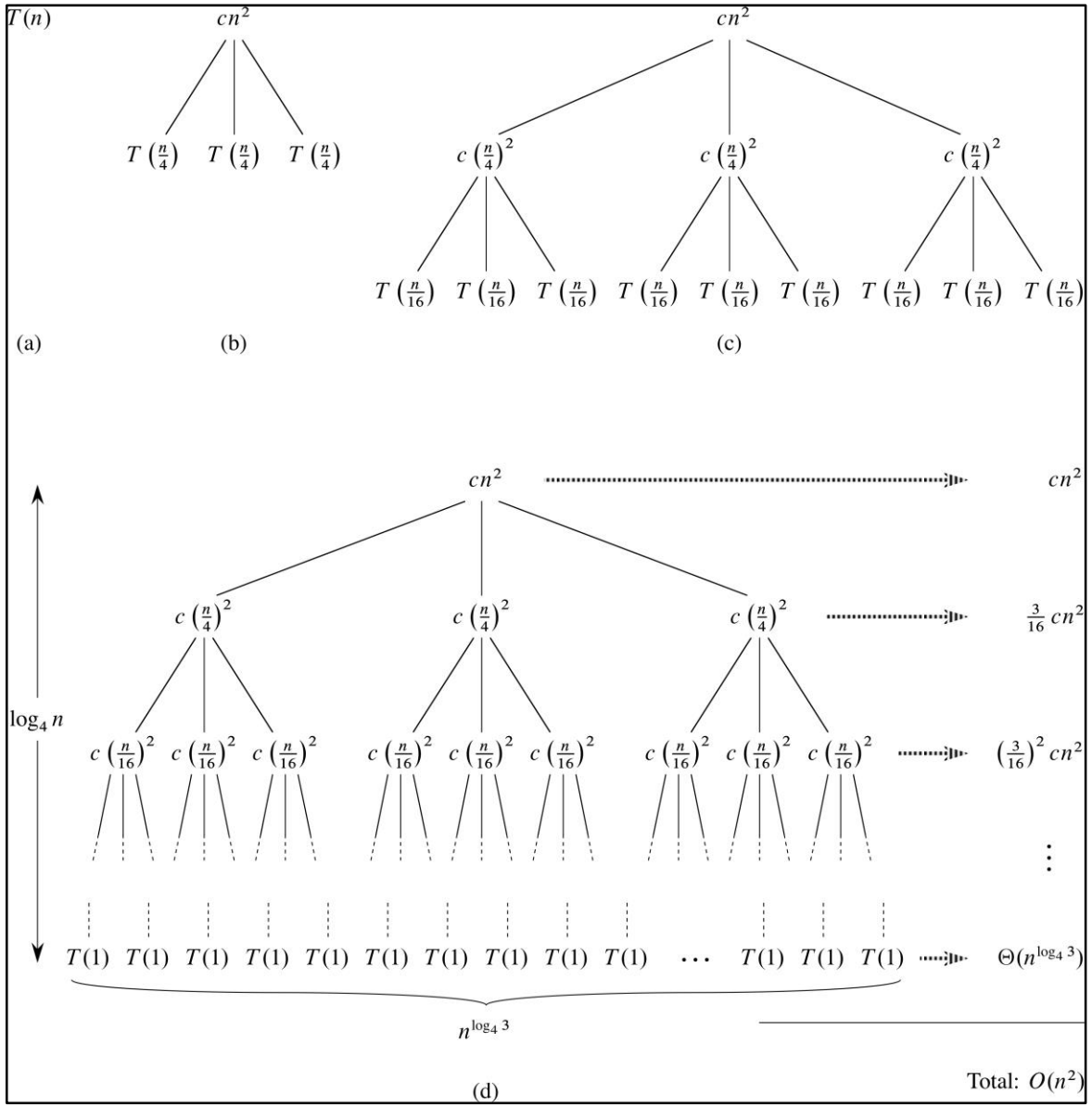


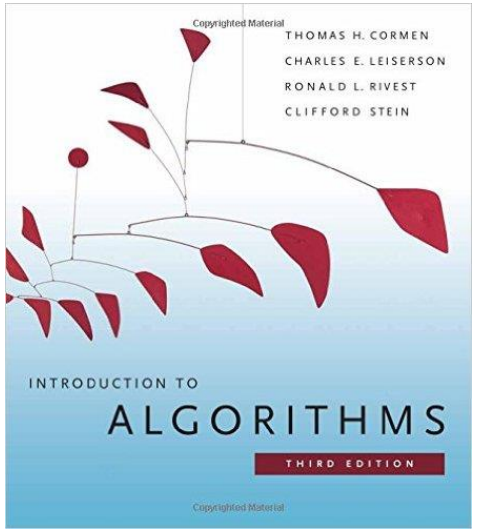
THE RECURSION TREE METHOD– NOT FOR TEST

- In a ***recursion tree***, each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.
- We sum the costs within each level of the tree to obtain a set of per-level costs, and then we sum all the per-level costs to determine the total cost of all levels of the recursion.
- Recursion trees are particularly useful when the recurrence describes the running time of a divide-and-conquer algorithm.
- A recursion tree is best used to generate a good guess, which is then verified by the substitution method
- When using induction, you can often tolerate a small amount of "sloppiness"
- For example, $T(n) = 3\,T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$ can we written out as $T(n) = 3\,T(n/4) + cn^2$



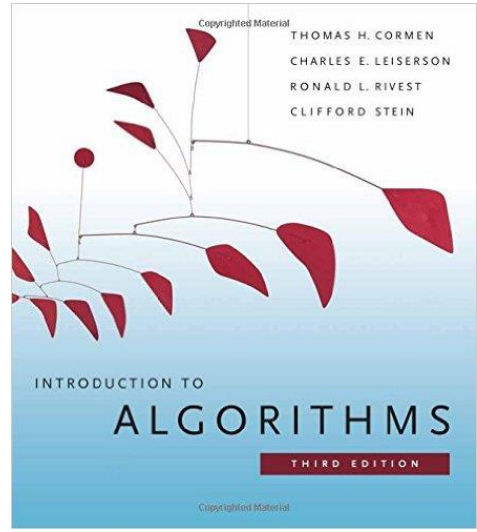
THE RECURSION TREE METHOD– NOT FOR TEST





THE RECURSION TREE METHOD– NOT FOR TEST

- $n, \frac{n}{4}, \frac{n}{4^2}, \ldots, \frac{n}{4^k}$. Base case is $\frac{n}{4^k} = 1$ hence $k = \log_4 n$. Thus the tree has $\log_4 n + 1$ levels.



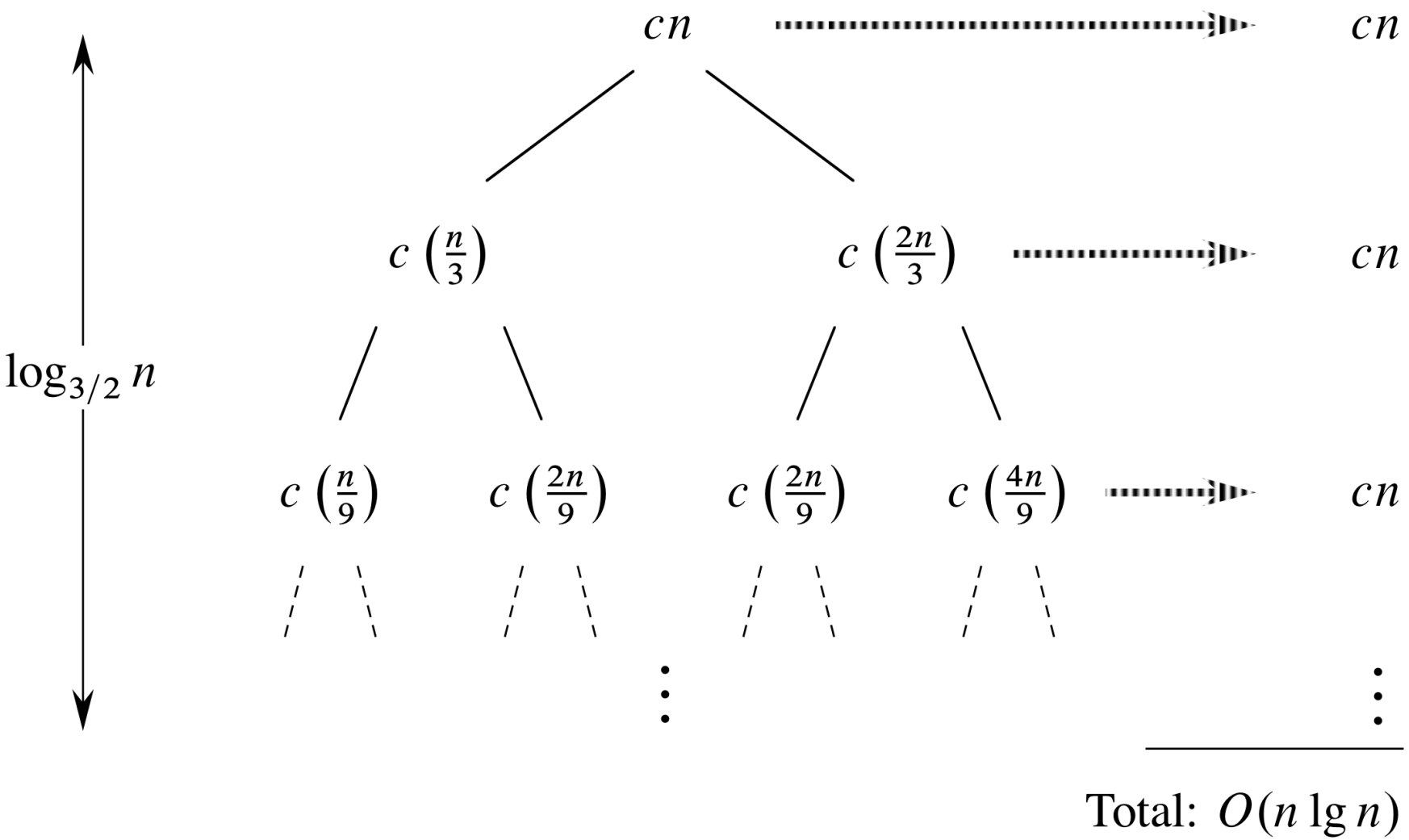
- The cost at each level: $cn^2, \frac{3}{16}cn^2, \left(\frac{3}{16}\right)^2 cn^2, \ldots, \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + 3^{\log_4 n} T(1)$
- Hence we have: $cn^2, \frac{3}{16}cn^2, \left(\frac{3}{16}\right)^2 cn^2, \ldots, \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$
- Total (see geometric series):

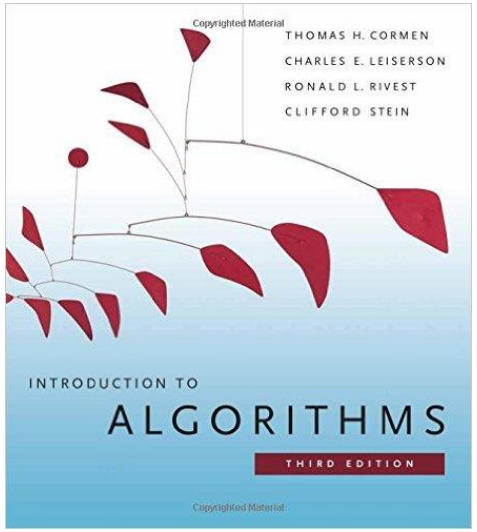
$$cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) < \frac{1}{1 - \left(\frac{3}{16}\right)} cn^2 + \Theta(n^{\log_4 3}) = O(n^2)$$

- Why is it the case that in this example we also have: $\Theta(n^2)$

THE RECURSION TREE METHOD 2– NOT FOR TEST

- Find the recursion tree for $T(n) = T(n/3) + T\left(\frac{2n}{3}\right) + O(n)$





THE MASTER METHOD – NOT FOR TEST

- provides a "cookbook" method for solving recurrences of the form:

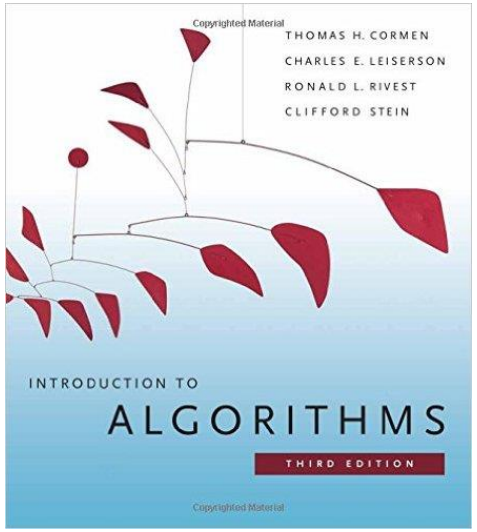
$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- where a ≥ 1 and b > 1 are constants and f (n) is an asymptotically positive function.

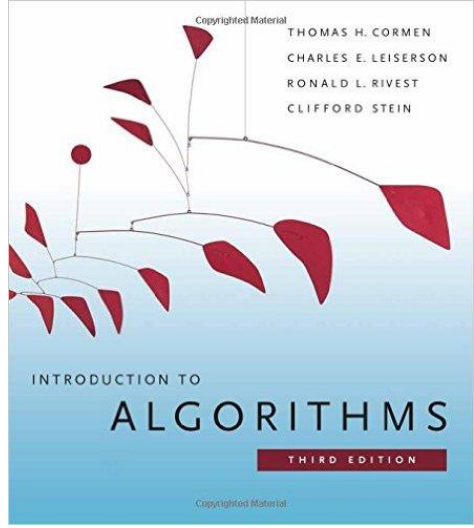
- The master method requires memorization of three cases, but then the solution of many recurrences can be determined quite easily

- an algorithm that divides a problem of size n into ***a*** subproblems, each of size ***n/b***, the cost of dividing the problem and combining the results of the subproblems is described by the function ***f(n)***

- For **mergesort**, what were ***a, b, f(n)***?
- What about **binary search**?



THE MASTER METHOD – NOT FOR TEST

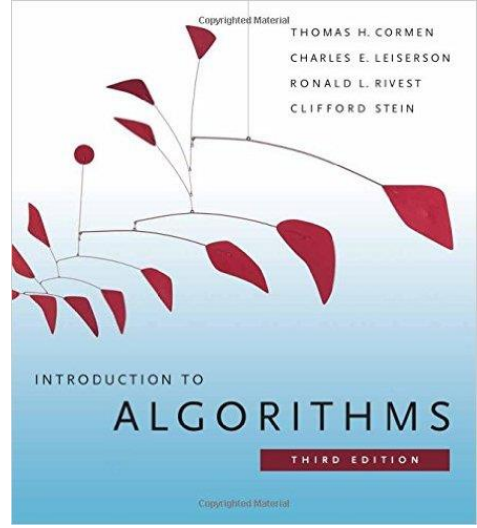


- **SIMPLIFICATION!!!**
  Let a ≥ 1 and b > 1 be constants, let f (n) be a function, and let T (n) be defined on the nonnegative integers by the recurrence
  $$\boldsymbol{T(n) = aT(n/b) + f(n)},$$
  where we interpret n/b to mean either ⌊n/b⌋ or ⌈n/b⌉. Let $f(n) = \Theta(n^c)$

- If $c < \log_b a$ (using big-Oh notation) then $T(n) = \Theta\left(n^{\log_b a}\right)$
- If $c = \log_b a$ then $T(n) = \Theta\left(n^{\log_b a} \log n\right)$
- If $c > \log_b a$ then $T(n) = \Theta(n^c)$ ("regularity" condition skipped)

- Note: we compare the function f(n) with $n^{\log_b a}$.
  The larger of the two determines the solution. If equal, we multiply by a logarithmic function

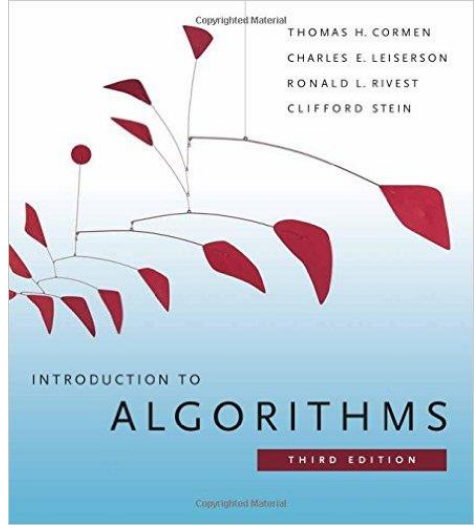
THE MASTER METHOD – NOT FOR TEST



Algorithm	Recurrence Relationship	Run time
Binary search	$T(n) = T\left(\frac{n}{2}\right) + O(1)$	$O(\log n)$
Binary tree traversal	$T(n) = 2T\left(\frac{n}{2}\right) + O(1)$	$O(n)$
Optimal Sorted Matrix Search	$T(n) = 2T\left(\frac{n}{2}\right) + O(\log n)$	$O(n)$
Merge Sort	$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$	$O(n \log n)$

- Source: http://www.cse.unt.edu/~tarau/teaching/cf1/Master%20theorem.pdf

THE MASTER METHOD – NOT FOR TEST



- Find the running time for $T(n) = 9T\left(\frac{n}{3}\right) + n$
- Find the running time for $T(n) = T\left(\frac{2n}{3}\right) + 1$
- Find the running time for $T(n) = 3T\left(\frac{n}{4}\right) + n\log n$

ANALYZING DIVIDE-AND-CONQUER ALGORITHMS

Master Theorem (chapter 4):

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c, \\ aT(n/b) + D(n) + C(n) & \text{otherwise.} \end{cases}$$

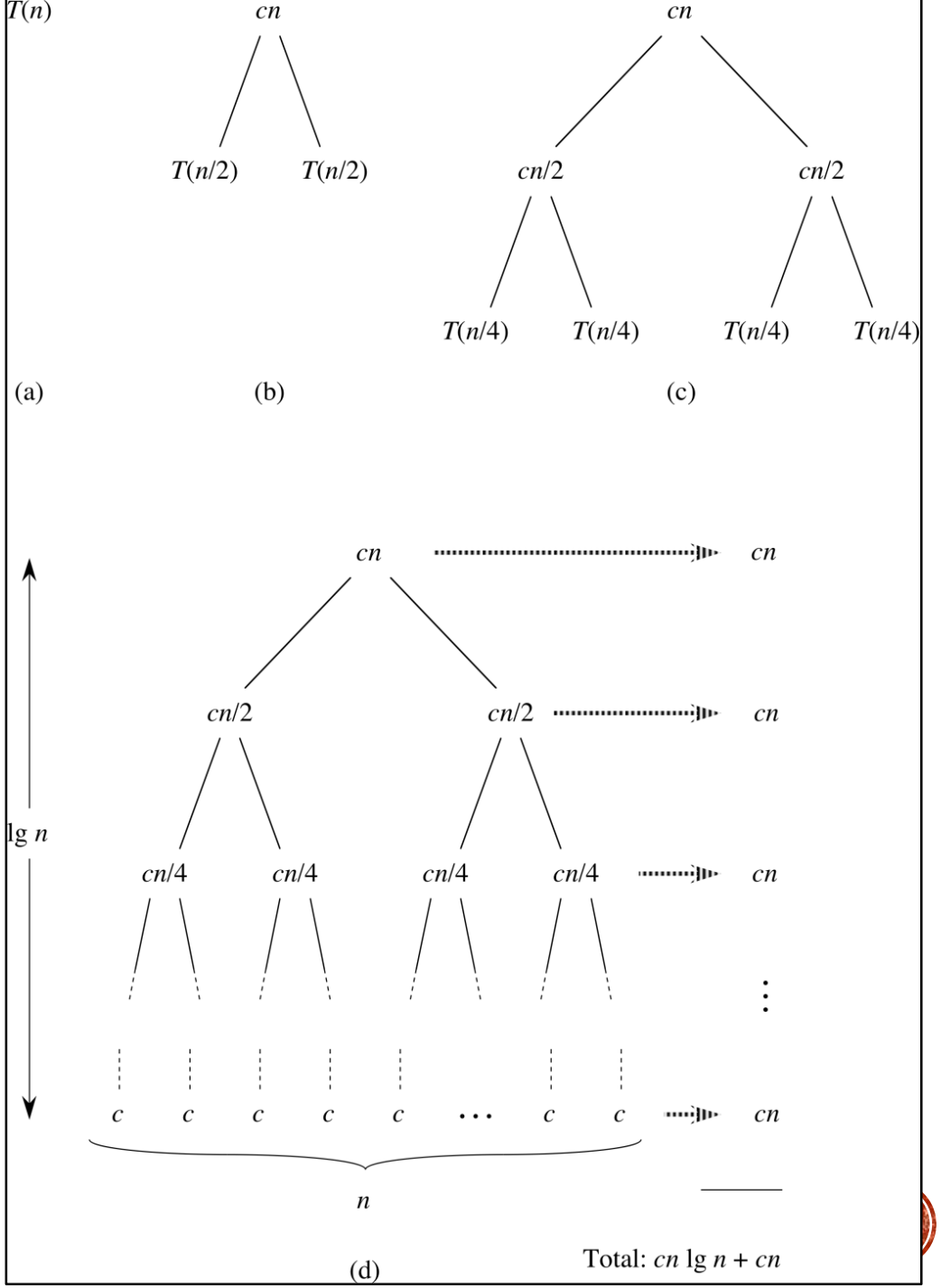
- Suppose that our division of the problem yields a subproblems, each of which is $1/b$ the size of the original. (For merge sort, both a and b are 2. If we take $D(n)$ time to divide the problem into subproblems and $C(n)$ time to combine the solutions to the subproblems into the solution to the original problem, we get the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

Merge Sort:

- divide step just computes the middle of the subarray, takes constant time. Thus, $D(n) = \Theta(1)$.
- recursively solve two subproblems, each of size $n/2$, contributes $2T(n/2)$ to the running time
- Merge takes time $\Theta(n)$, so $C(n) = \Theta(n)$
- Using master method (…!) $T(n)$ is $\Theta(n \lg n)$, where lg n stands for $\log_2 n$

WITHOUT MASTER METHOD



HOMEWORK FOR MODULE 2

DUE: see moodle, 11:59 pm

Write only one C# program that contains all three parts below

Write one C# program that includes all of the following:

- [15 points] Modify the bubble sort algorithm seen in class so it works with an array of strings AND the array will have the values sorted in reverse. Also add a local variable count (use long count) of the number of comparisons that were performed. Display it before exiting this method. Call it: **static void bubbleReverseSort(string[] arr)**
- [15 points] Modify the selection sort algorithm seen in class so it works with an array of strings AND the array will have the values sorted in reverse. Also add a local variable count (use long count) of the number of comparisons that were performed. Display it before exiting this method. Call it: **static void selectionReverseSort(string[] arr)**
- [15 points] Modify the merge sort algorithm seen in class so it works with an array of strings AND the array will have the values sorted in reverse. . Also add a local variable count (use long count) of the number of comparisons that were performed. Display it before exiting this method. Call it: **static void mergeReverseSort(string[] arr)**
- [15 points] Modify the quick sort algorithm seen in class so it works with an array of strings AND the array will have the values sorted in reverse. . Also add a local variable count (use long count) of the number of comparisons that were performed. Display it before exiting this method. Call it: **static void quickReverseSort(string[] arr)**
- [40 points] In **Main** read the entries given in a **input.txt** (one line per entry) and store them into an array. Make four copies of it. Then, on each copy call **bubbleReverseSort, selectionReverseSort, mergeReverseSort, mergeReverseSort**. For each of these measure the execution time (how long it took to run the reverse sorting) and display this time.
  Hint for measuring execution time: ➔

```
var watch = System.Diagnostics.Stopwatch.StartNew();
// the code that you want to measure comes here
watch.Stop();
var elapsedMs = watch.ElapsedMilliseconds;
```

- **Reminder: Add a comment at the beginning of EACH method (including Main()) in which you include the running time of the method.**
  - **10% of this homework grade will go to correctly identifying the time complexity of each of your methods.**
- **Reminder: If your code does not compile, crashes at start, or contains no meaningful comments, it will automatically be graded with 0!**