Questions and/or Exercises to work out and turn in:

Grading Guidelines (See Appendix):

A right answer will get full credit when:

1. It is right (worth 25%)
2. It is right **AND** neatly presented making it easy and pleasant to read. (worth an **extra** 15%)
3. There is an **obvious and clear link** between 1) the information provided in the exercise and in class and 2) the final answer. A clear link is built by properly writing, justifying, and documenting an answer (worth an **extra** 60%).
4. Calculation mistakes will be minimally penalized (2 to 5% of full credit) while errors on units will be more heavily penalized.

You are welcome/encouraged to discuss exercises with other students or the instructor. But, ultimately, **personal** writing is expected.

* USE THIS FILE AS THE STARTING DOCUMENT YOU WILL TURN IN. **DO NOT DELETE ANYTHING FROM THIS FILE:** JUST **INSERT** EACH ANSWER **RIGHT AFTER ITS QUESTION/PROMPT**.
* IF USING HAND WRITING (STRONGLY DISCOURAGED), **USE THIS FILE** BY CREATING SUFFICIENT SPACE AND WRITE IN YOUR ANSWERS.
* FAILING TO FOLLOW TURN IN DIRECTIONS /GUIDELINES WILL COST **A 30% PENALTY.**

Objectives of this assignment:

* to use and manipulate the concepts presented in this module
* to use and manipulate the definitions of O(g(n)), Ω(g(n)), and Θ(g(n))
* to get familiar with the “order” of usual functions: polynomials, square root, logarithms, exponentiatl...

What you need to do:

Answer the questions and/or solve the exercises described below.

Exercise (100 points) Divide Search (Read the Appendix about how to get full credit)

Consider the algorithm DichotomySearch(A, p, r, x): the description of this algorithm is provided below.

Inputs:

* a sorted array *A*
* index *p* of the first element
* index *r* of last element
* an element *x* to search

Output:

* index of element *x* in Sequence *A* if x exists in *A*
* -1 if *x* does not exist in Sequence *A*.

Algorithm description

int DichotomySearch(A, p, r, x)

        if (r >= p)

            midpoint = p + (r-p)/2;

             if A[midpoint] == x

               return midpoint;

            if A[midpoint] > x

               return DichotomySearch(A, p, midpoint-1, x);

else

             return DichotomySearch(A, midpoint+1, r, x);

        return -1;

The objective of this exercise is to derive the time complexity (running) time of the *Faster* search.

1. (12 points) Let A = (3, 7, 13, 15, 25, 29, 30, 38, 45, 58, 65, 70). Assume that the index of the first element is 1.
   1. Execute manually DichotomySearch(A,1, A.length, 66). What is the output? (**Provide the indices and contents of** the cells checked/searched during the search)

First I will just say from immediate look that 66 is not in the given array A, so this will definitely return -1, like above in the algorithm described. A.length will be equal to 12 since the first value will start at 1. Here now are the steps to manually execute DichotomySearch(A,1, A.length, 66) :

* Initial p = 1, r = 12
* Midpoint = 6 (12/ 2 = 6)
* A[midpoint] = 29 which is less than 66 and not equal to
* Recursive call on the right half, DichotomySearch(A, 7, 12, 66)
* Midpoint = 9
* A[midpoint] = 58 which is less than 66 and not equal to
* Recursive call on the right half, DichotomySearch(A, 10, 12, 66)
* Midpoint = 11
* A[midpoint] = 70 which is greater than 66 and not equal to
* Recursive call on the left half, DichotomySearch(A, 10, 10, 66)
* Midpoint = 10
* A[midpoint] = 65 which is less than 66 and not equal to
* Recursive call on the right half, DichotomySearch(A, 11, 10, 66)
* Since r < p, return -1
  1. Execute manually DichotomySearch(A,1, A.length, 15). What is the output? **Provide the indices and contents of** the cells checked/searched during the search)

Executing DichotomySearch(A, 1, A.length, 15):

The output will be 4, as 15 is found at index 4 in the array A.

Indices and contents checked during the search:

1. Initially, we set p = 1 and r = 12, indicating the start and end indices of the array A.

2. We calculate the midpoint as 12 / 2 = 6. So, the midpoint is at index 6.

3. We check the element at the midpoint, A[6], which is 29. Since 29 is greater than 15, we know that the element we are searching for 15 must be in the left half of the array.

4. We update the range to search within the left half. Now, p = 1 and r = 6.

5. Recursively, we calculate the midpoint of this new range as 6 / 2 = 3.

6. We check the element at the new midpoint, A[3], which is 15. Since 15 is equal to the element we're searching for, we return the index of the midpoint, which is 4.

* 1. Execute manually DichotomySearch(A,1, A.length, 38). What is the output? **Provide the indices and contents of** the cells checked/searched during the search)

Executing DichotomySearch(A, 1, A.length, 38):

The output will be 8, as 38 is found at index 8 in the array A.

Indices and contents checked during the search:

1. Initially, we set p = 1 and r = 12, indicating the start and end indices of the array A.

2. We calculate the midpoint as 12 / 2 = 6. So, the midpoint is at index 6.

3. We check the element at the midpoint, A[6], which is 29. Since 29 is less than 38, we know that the element we are searching for (38) must be in the right half of the array.

4. We update the range to search within the right half. Now, p = 7 and r = 12.

5. Recursively, we calculate the midpoint of this new range as (7 + 12) / 2 = 9.

6. We check the element at the new midpoint, A[9], which is 58. Since 58 is greater than 38, we know that the element we are searching for (38) must be in the left half of the current range.

7. We update the range to search within the left half. Now, p = 7 and r = 8.

8. Recursively, we calculate the midpoint of this new range as (7 + 8) / 2 = 7.

9. We check the element at the new midpoint, A[7], which is 38. Since 38 is equal to the element we're searching for, we return the index of the midpoint, which is 8.

1. (2 points) Which operation should you count to determine the running time T(n) of the *DichotomySearch* in a sequence A of length n? To determine the running time T(n) of DichotomySearch in a sequence A of length n, we count the number of comparisons made during the execution of the algorithm. The running time could then be consider the Big O of log(n).
2. (30 points) **Let us COUNT** these comparisons only (if (r >= p, (if A[midpoint] == x), and (if A[midpoint] > x) ) to express the running time T(n) as a recurrence relation. Make sure to explain each coefficient/constant/variable you use. Provide the value of the constants if any. Make sure to tie the expression(s) to the algorithm you are analyzing. Use the steps used on Slide M4:14. On the algorithm below label what you are counting just like what we did on Slide M4:14. Then, derive the recurrence relation.

int DichotomySearch(A, p, r, x)

        if (r >= p)

            midpoint = p + (r-p)/2;

             if A[midpoint] == x

               return midpoint;

            if A[midpoint] > x

               return DichotomySearch(A, p, midpoint-1, x);

else

             return DichotomySearch(A, midpoint+1, r, x);

        return -1;

If we are to use these three comparisons only, I will dictate each to be c1 c2 c3. T(n) represents the running time of the algorithm on an input of size n. Comparison 1 is performed once in each recursive call when the search range is valid, so it contributes c1. Comparison 2 and Comparison 3 are also performed once in each recursive call when the search range is valid, so they contribute c1 and c2 respectively as well.

The recursive calls occur only when the search range is valid, so the recurrence relation becomes:

T(n) = T(n/2)+ c1 + c2 + c3

This recurrence relation captures the time complexity of the DichotomySearch algorithm by considering only the comparisons made during its execution.

1. (24 points) Solve the recurrence relation T(n) (found in Question 3) using the recursion-tree method. Justify your answer following the steps and information shown on Slide M4:21. We must see the same information: columns, tree.....

Level 1 or the root will cost c times n. This represents the initial call with the input size of n. Then at level 2 the two recursive calls with input size n/2. This idealized means that each call at this level has a cost of c times (n/2). Level 3 has 4 recursive calls with the input size being n/4. This again idealized means that each call has a cost of c times (n/4).

The recursion tree will have a total of log2(n) + 1 levels, as we're dividing the input size by 2 at each level until reaching the base case.

Level 1 (Root): T(n)

|

|

Level 2: / \

T(n/2) T(n/2)

/ \ / \

/ \ / \

Level 3: T(n/4) T(n/4) T(n/4) T(n/4)

.....

Level i: T(n/2^i) T(n/2^i) ... T(n/2^i)

|

|

|

Level log2(n): T(1) // This should total 1.

A math equations and formulas

Description automatically generated

1. (20 points) Solve the recurrence relation T(n) (found in Question 3) using the substitution method

Justify your answer following the steps and information shown on Slide M4:24-25.

A white text with black text

Description automatically generated

1. (12 points) Solve the recurrence relation T(n) (found in Question 3) using the master method

Justify your answer following the steps and information shown on Slide M4:30-32.

**A white background with black text

Description automatically generated**

**Appendix**: Grading: What is an OBVIOUS and CLEAR LINK?

Here is an example to explain what an **obvious and clear link** is and how we grade your work.

Consider the following problem:

"(100 points) John travels from Auburn to Atlanta in his car at a speed of 60 mph. Leaving at 8am, at what time will John reach Atlanta".

Here are the answers of three students and their scores:

* **Student 1** answers: "9:48am". Student 1 will get 25 points.
* **Student 2**answers : "John will reach Atlanta at 9:48am". Student 2 will get 25+15 = 40 points
* **Student 3** answers: "The time t to travel a distance d at speed v is equal to d/v = d/60mph. The problem does not provide the distance d from Auburn to Atlanta. Based on GoogleMaps, the distance from Auburn to Atlanta is approximately 108 miles (**document is attached**).



Therefore, the time t = 108 miles/60mph \* 60 minutes/hour= 108 minutes. Since John left at 8am, he will then reach Atlanta at 8am + 108 minutes = 8 am + 60 minutes + 48 minutes = 9:48".

**Student 3** will get 25 + 15 + 60 = 100 points

Do you see the **direct** **link** going from the data provided in the question to the final answer, using general knowledge/formula and documents?.... Can you now solve the following problem and get 100 points?

"(100 points) Alice travels from Auburn to Atlanta in her car at a speed of 60 mph. Leaving at 8am, at what time will Alice reach Atlanta assuming that she had a flat tire that delayed her 30 minutes".