DSA Set Assignment 1

## Arrays and Searching

1. What are the main advantages and limitations of arrays? (5 points)

|  |  |
| --- | --- |
| Advantages | Limitations |
| Fast access to elements (by index) | Fixed size that you need to know before defining |
| Reduced memory usage | Contiguous memory allocation |
| Hold same type of elements | Hold the same type of elements |
| Easy to make | Insert and delete process complexity |

1. Answer all parts of this question using this array of 16 elements, all of which are whole positive numbers:

11, 23, 9, 99, 18, 3, 209, 874, 511, 1, 99, 312, 256, 319, 472, 0

* 1. Using SMC notation, list the notation and describe the operations required to find an element in the sorted array using a binary search. (Assume the list is already sorted) (20 points)

Set n = 16

Set element – 256 //element we need to find

Set a[n] = [0, 1, 3, 9, … 874] //sorted array

Set left = 0 //left and right edge of searching

Set right = Sub(Get n, 1)

Lable FindElement:

Compare (Get left < Get right)

Set middle = Div(Get left + Get right, 2)

Compare (Get a[middle] == Get element) Return middle

Compare (Get a[middle] < Get element) Set left = Add(middle, 1)

Compare (Get a[middle] > Get element) Set right = Sub(middle, 1)

If element was found it is it, if not element was not found

## Sorting and Big O

1. What does Big O notation represent? (5 points)

It describes the connection between the size of an input and complexity of using it. It helps analyse the worst performance of an algorithm.

1. What is the best-case performance of any algorithm, as described using Big O? (5 points)

O(1) - constant (doesn't depend on the size)

1. Using SMC notation, describe the operations required to count the occurrences of a value in an *un*sorted array if it is present. (10 points)

Set n = 8 //length

Set a[n] = [1, 2, 2, 10, 7, 4, 5, 4]

Set count = 0

Set i = 0

Set value = 4

Lable CountValue:

Compare (Get a[i] == value)

Set count = Add (count, 1)

Set i = Add(i, 1)

Compare (Get i != Sub(Get n, 1))

Jump CountValue

Return count

1. What is the time complexity of the algorithm from question 5? (15 points)

O(n). It goes through each value of an array which makes it linear search

1. Using pseudocode or a real programming language of your choice (C, C++, C#, Python), write a simple function that adds a new element to the array, so that the array remains sorted, e.g, *‘int insertNewElement(int x)*. (20 points)

void InsertNewElement (int x, int a[], int maxSize, int size):

IF size == maxSize:

alert “No more space left in this array”

return

int newPosition

FOR each number from array:

IF number is less then x:

increase newPosition by 1

ELSE:

newPosition = number’s position //for example in C++ there is a function find (element) which returns the index

FOR each number which position is greater than new position:

Position in array increases by 1 //shifting all elements before insertion

a [new Position] = x

## Linked Lists and ADT

1. What does an Abstract Data Type ‘node’ represent? (5 points)

Usually it's a two-value component which contains: it's data and a link to the next node(s)

1. What is the Big O performance of a search through a Linked list? Note: it’s the same for a singly- or doubly-linked list. (5 points)

O(n). You need to go from the first node to (the worst case) last one. Linear complexity

## Numerical Algorithms

1. Using correct and detailed pseudocode, write two functions that each take an integer input and then output the calculation of the “sum of n powers of 2”.
   1. Without using a maths function that performs exponentiation, function (a) should return the result of the formula: sum = 2(n+1) - 1 (25 points)

Int n = 7

Int sum = 0

Int powered = 1

For i = 0; i <= n; i ++:

For y = 0; y < i; y ++:

powered = powered \* 2

sum = sum + powered

* 1. Function (b) should perform the lengthy iterative calculation to reach a sum. (25 points)

Int n = 7

Int sum = 0

Int Powered (int n):

If (n = 0) return 1;

sum += Math.power(2, n -1)

return sum

1. What is the Big O performance of each of the functions in question #10? (10 points)

a. O(2n). Runs nearly twice n size

B. O(n). Runs only n amount of times

## Stack Queues and Trees

1. What is the simple acronym that describes a Stack’s function? (5 points)

LIFO (Last in First Out)

1. What are the basic operations that a Queue must support? (5 points)

* Peek (top) look at the top element
* Push (enqueue) inserts an element to the back
* Pop (dequeue) removes an element at the front
* isEmpty

1. For all parts of this question, use the following set of data elements as the source data: {“Thor”, “Zeus”, “Isis”, “Mars”, “Lugh”, “Anu”}
   1. Apply these operations in sequence to a Stack structure, iterating through the source data elements as input: Push, Push, Pop, Push, Pop, Push. After all operations are finished, what data will the Stack contain? (10 points)

|  |  |
| --- | --- |
| 1 | Mars |
| 0 | Thor |

* 1. Apply these operations in sequence to a Queue structure, iterating through the source data elements as input: Push, Push, Pop, Push, Pop, Push. After all operations are finished, what data will the Queue contain? (10 points)

|  |  |
| --- | --- |
| 1 | 0 |
| Mars | Isis |

* 1. Iterate through all the source data elements in order and add them to a Tree structure. If the first is added to the root node, and each node in the Tree is allowed only 2 child nodes, what will the resulting tree look like? (20 points)

Thor

/ \

Zeus Isis

/ \ /

Mars Lugh Anu

* 1. Which of these data structures is the best choice to allow the most flexible access to any element? Describe why for full marks. (10 points)

A tree. Searching through a stack or a queue is good for step-by-step using whereas different tree types (binary, k-d tree) can give you more flexibility and accessibility.

Maps and Trees

360 : “degrees”

45 : “minutes”

10 : “seconds”

99 : “balloons”

9 : “lives”

101 : “dalmatians”

1. Answer all parts of this question using this set of data consisting of key-value pairs, where the key is an integer, and the value is a string.
   1. What are two good choices for data structures to store the string data so each element can be uniquely accessed using its key? Justify your choices with a full explanation. (10 points)

It could be dictionary (map) or a tree. Additionally, it could also be a hash table (each element has its own unique key that goes through hashing (difference from map)).

Map is one of the best ways to store key-value pairs. Elements are easy to access and use.

Tree can efficiently store data by key which will make access and search easier and faster

* 1. Using the encoding approach of a Radix Tree, create a Tree diagram to show how key fragments can be used to identify the edges between nodes. (25 points)

All numbers can be turned into binary, so that we’ll have “prefixes” to define and connect keys.

A close up of writing on a wall

Description automatically generated

* 1. What is the Big O performance of a search through the tree from question #3b? (15 points)

O(l), where l is a length of a search

## Recursion

1. In your own words, describe the 3 rules that recursive functions must follow. (10 points)

* There should be at least one case where recursion ends
* Make recursive calls with smaller data only
* It should handle all valid inputs

1. Answer all parts of this question using the tree diagram below as sample data, and assuming the number in each node is a point value. Your solutions should be generic to work with any data set; test them using this tree. A diagram of numbers and circles

   Description automatically generated
   1. Using detailed and precise pseudocode or a programming language of your choice, write a recursive algorithm to find the **path** from the root to the leaf node with the **largest total value**. Your solution should consist of one or more recursive functions, plus any additional functions as needed. (40 points)

Vector<int> FindPathWithLargestValue(Node node, currentPath, largestValuePath, currentValue, largestValue):

If (node == null) return; //end of recursion

currentPath.push(node->value)

currentValue += node->value

if (node->left == null && node->right == null) //if we got to the leaf

if (largestValue < currentValue)

largestValue = currentValue

largestValuePath = currentPath

else //go to the next branch, both left and right

FindPathWithLargestValue(node->left, currentPath, largestValuePath, currentValue, largestValue)

FindPathWithLargestValue(node->right, currentPath, largestValuePath, currentValue, largestValue)

currentPath.pop() //removes leaf from path to go back in tree

return largestValuePath

int main()

vector<int> largestValuePath, currentPath

Node node = Node

largestValuePath = FindPathWithLargestValue(node, currentPath, largestValuePath, 0, 0)

class Node()

int value

Node left

Node right

## Graphs

Complete all questions referring to this website: <https://oracleofbacon.org/movielinks.php>

Using the software tool of your choice (Visio, draw.io, etc.), create a diagram of the graph showing relationships of these actors to Kevin Bacon

* Tom Hardy
* Helen Mirren
* Sacha Baron Cohen
* Kate Winslet

1. Include at least one relationship using multiple paths (5 points)

See answer for question 19

1. Add **edge attributes** to include film names and year (5 points)

A diagram of a company

Description automatically generated

1. Create the definitions for data structure(s) that could be used to correctly to describe this graph in a program, including the various data elements. Note: this question is asking for data structures in the programming language sense, e.g., “struct Node” and “struct Edge”. (15 points)

I can think of creating a class (or a struct) Actor which will hold their name and a vector of Struct Film (where the name, the year of the film and an array of actors will be stored)

Also, database can be created table for actors (PK + name), films (PK + name + year), films that actors were in (junction table with actor FK, film FK)

Tree structure is not the best solution here since its idea is that it doesn't have loops in it which might appear with many to many relations