**ENSF 338**

**Practical Data Structures and Algorithms**

**Assignment 2**

**Group no. – 12**

**Section - 2**

**Group members:**

**Akshpreet Singh**

**Rohil Dhillon**

**Work performed by each member:**

Exercise 1: Rohil

Exercise 2: Aksh

Exercise 3: Rohil

Exercise 4: Rohil

Exercise 5: Aksh

**GitHub Repository:**

<https://github.com/Rohil1710/Assignment2_338.git>

**Exercise 1:**

Ans.1.1) Memoization in general terms is a method/technique used to improve the efficiency of a process by storing results for calculations done previously and the inputs that led to that particular result. When the same inputs pop up again we reuse the stored result.

Ans.1.3) The given code is a recursive function that produces the Fibonacci sequence.

Ans.1.4) The given code is not an example of the divide-and-conquer algorithm because it does not generate problems that are smaller and smaller to solve. The given code generates multiple problems that are just as complex as the original with the number of computation increasing exponentially.

Ans.1.5) The complexity of the given code is O() i.e. exponential time complexity.

Ans.1.6)

def func(n, dict={}):

    if n == 0 or n == 1:

        return n

    if n in dict:

        return dict[n]

    else:

        result = func(n-1, dict) + func(n-2, dict)

        dict[n] = result

        return result

Ans.1.7) Performing the analysis of the code optimized for memorization:

Since the code stores values for “func(n)” that have been previously calculated in the dictionary “dict”, the code only has to produce a result for each value of n once. The time required to evaluate the function and store takes O(1) time. Therefore, if we evaluate the function and store the result n times then the time complexity of the optimized code is O(n).

Ans.1.8) Plots are below: Chart

Description automatically generated

Chart, histogram

Description automatically generated

Ans.1.9)

* The plot for the optimized code with memoization has a linear slope (though negative and positive) which means that its time complexity is O(n).
* The plot for the unoptimized code is non-linear and grows exponential and seems to be quadratic therefore its time complexity is O(n^2).

As we can see the time complexities we calculated and the time complexities observed from the plots are the same. This verifies our results.

**Exercise 2:**

**Exercise 3:**

Ans.3.1) Some key aspects that make interpolation search better than binary search are:

* It gets to the result faster compared to binary search because it doesn’t star dividing the uniform data simply at the middle. Instead we divvy up the data from an approximate location of where we think the required item is.
* The time complexity of interpolation is O(log(log(n))) compared to O(log(n)) for binary search assuming the data being searched through is uniform. Therefore, even in the worst case, interpolation is better than binary search.

Ans.3.2) Yes, performance will be affected. The way interpolation search was developed assumed uniform data which means the difference between adjacent elements is the same throughout. Because of this property of uniform data we can use the formula approximate the location of the required data. But, if we use data that is distributed differently the data items may not be separated equally which will cause the efficiency of the interpolation search to drop. This is because the accuracy of the formula to predict the approximate location of the required item will decrease.

If the data is for instance normally distributed then values of the items are most likely closer to the mean than father away, this can cause the interpolation search to take longer to find the required value.

Ans.3.3) If we want to modify the interpolation search to follow a different distribution then we have to change the “pos” variable which approximates the location of the desired element. For example to adapt the “pos” variable to work with the normal distribution we would need to incorporate the mean, the standard deviation, and the cumulative distribution function of the normal distribution.

Ans.3.4.a) Linear search is the only choice to use for searching through data when the data given is unsorted as both binary search and interpolation search require sorted data.

Ans.3.4b) A case where linear search may outperform binary and interpolation search is when the array is particularly small in size because it takes a considerable amount of time to sort data and then search through it using binary or interpolation search.

In terms of improving binary and interpolation search we can use interpolation or binary to narrow our search area in the data to a small number of elements and then sequentially search through the remaining elements.

**Exercise 4:**

Ans.4.1) The advantages of an array compared to a linked list are:

1. We have quick access to all elements in the array because they’re stored in contiguous memory locations, the time complexity of accessing an element is O(1).
2. Arrays also use less memory than linked lists as they don’t require additional memory for pointers to each other.

The disadvantages of an array compared to a linked list are:

1. An array is of fixed size and cannot be contracted or expanded whereas a linked list can be.
2. Insertions and deletions are relatively inefficient compared to a linked list. This is because insertions and deletions in a linked list only require updating pointers which may take as little as O(1) time.

Ans.4.2) The function can be implemented as follows:

* Go to the given index and from that point onwards shift all elements by one to the left (deletion). (O(n))
* After shifting to the left, shift all elements at that point one to the right. (O(n))
* Then with the empty slot at the point place the element provided (insertion). (O(1))

Ans.4.3)

a.) Selection sort cannot be used in a linked list because we need to move back and forth which in a linked list is not possible but is in an array. Using this method of sorting in an array results in a time complexity of O().

b.) Insertion sort can be used in both linked lists and arrays as it simply goes through the data structure and builds a sorted version of the data structure we already had. Moreover, because insertion functions the same in both arrays and linked list it has the same time complexity which is O().

c.) Merge sort can be used in both linked lists and arrays because we can divide both linked lists and array into smaller linked lists and smaller arrays. Time complexity of merge sort in both linked lists and arrays is O(n\*log(n)).

d.) Bubble sort can be used in a linked lists and an array. In a linked list it will iterate through all the elements and swap nodes using pointers. It has a time complexity of O().

e.) Quick sort can be applied to linked but we have to store the pointers to the head and tail of linked list to merge the elements in the right order. While using merge sort with linked lists we use the pivot to divide the linked list into sublists. It has a time complexity of O().