CW1 Algorithms (COMP0005)

Student number: 19055854

Question 1

1.a) The complexity of an unsuccessful search for the given algorithm is its asymptotically worst case, constant $O(n)$. Unsuccessful search implies the absence of the 'key' or a particular element in a given array $A[0..n-1]$. Subsequently, the number of comparisons made is equivalent to the length of the array. The algorithm will compare each element $A[i]$, iterating $i = 0$ to $n - 1$ to 'key', hence the expected number of comparisons made is n, meaning the time complexity of the given unsuccessful search is $O\left(\frac{n+1}{2}\right)$, however the constants are ignored due to its insignificance at large numbers of n, hence O(n).

1.b) The asymptotic time complexity (worst case) for a successful search for the given algorithm is when the element that is searched for is the last element of the array $A[0..n-1]$. In that case, the algorithm would need to make $\frac{n+1}{2}$ comparisons, with the last element resulting in returning index i of the 'key'. Ergo, time complexity is $O\left(\frac{n+1}{2}\right)$ or just $O(n)$.

1.c) The average number of comparisons made is $\frac{n+3}{2}$, where n is the number of elements in array A.

Question 2

2.a) Yes, since,

$$log_2(2n) = log_2(2) + log_2(n)$$

$$log_2(2n) = log_2(n) + 1$$

$$log_2(n) + 1 \in O(log_2(n))$$

2.b) A similar statement would still be true for an arbitrary value of a:

$$log_a(2n) \in O(log_a(n))$$

Question 3

Given an array $A = \{0,0,0 \ldots 1,1, \ldots ,1,1\}$, such that all 0 appear before 1, the following pseudocode describes a search algorithm to return the smallest index of an element $A[i]$, such that $A[i] = 1$. The function $search(A, l, r)$ takes in the following parameters:

$$A = is\ the\ input\ array$$

$$left = marks\ the\ left\ serch\ boundary\ element, initially\ equals\ to\ 0$$

$$right = marks\ the\ left\ serch\ boundary\ element, initially\ equals\ to\ (A.length - 1)$$

```
SEARCH(A, left, right)
    if(right > left)
        int middle = (left + (right - 1))/ 2;
        if(A[0] == 1)
            return 0
        if(A[middle] == 1)
            if(A[middle] == 1 and middle == 0)
                return middle;
            else if(A[middle - 1] ==  0)
                return middle;
            else
                return SEARCH(A, left, middle);
        else
            return SEARCH(A, middle + 1, right);
    return - 1;
```

Calculating the time complexity of the SEARCH algorithm:

Length of the array after the first iteration $middle = \frac{n}{2}$, where n is the length of the array A. Subsequently, after some m iterations, the length of the array equates to $\frac{n}{2^m}$, which eventually, at some value of m becomes 1. Therefore,

$$n = 2^m$$

Thus,

$$m = \ log_2(n)$$

The "if", "else" statements at lines 4, 5, 7, 9, 11 all take constant time to execute. Conclusively, since the algorithm is a variation of the binary search algorithm, it follows that the asymptotic running time complexity of the SEARCH algorithm is $O(log_2(n))$. Full implementation of the given algorithm in Python language can be viewed below:

```
1  def SEARCH(A, left, right):
2      if(right > left):
3          middle = (left + (right - 1)) // 2
4          if(A[0] == 1):
5              return 0
6          if(A[middle] == 1):
7              if(A[middle] == 1 and middle == 0):
8                  return middle
9              elif(A[middle - 1] == 0):
10                 return middle
11             else:
12                 return SEARCH(A, left, middle)
13         else:
14             return SEARCH(A, middle + 1, right)
15     return -1
16
17 A = [0, 0, 1, 1, 1]
18 right = len(A)
19 left = 0
20
21 result = SEARCH(A, left, right)
22 if(SEARCH(A, left, right) == -1):
23     print("Unuccessful search")
24 else:
25     print("Element found at index", result)
26
```

Question 4

4.1) $500n + 100n^{\frac{3}{2}} + 50nlog_{10}n$ has complexity $O(n^{\frac{3}{2}})$

4.2) $nlog_3n + nlog_2n$ has complexity $O(nlog_2n)$

4.3) $n^2 log_2n + n(log_2n)^2$ has complexity of $O(n^2 log_2n)$

4.4) $0.5n + 6n^{\frac{3}{2}} + 2.5n^{\frac{7}{4}}$ has complexity of $O(n^{\frac{7}{4}})$

4.5) $0.01n + 100n^2$ has complexity of $O(n^2)$

4.6) $500n\, log_3n + 0.1n^2 + 200n$ has complexity of $O(n^2)$