Nathanael Paulemon

CSC 212

Assignment 4 Example One

Text highlighted in yellow are my answers.

**Assignment 4**

For each of the following functions code, calculate T(n), the time it takes to solve a problem of size n, and find the corresponding Big-O running time:

**Sequential search**

|  |  |
| --- | --- |
|  | def sequentialSearch(alist, item): |
| pos = 0 |
| found = False |
|  |
| while pos < len(alist) and not found: |
| if alist[pos] == item: |
| found = True |
| else: |
| pos = pos+1 |
|  |
| return found |

Because the while loop of the sequential search is executed at most n times. T(n) is equal to O(n).

**Bubble sort**

def bubbleSort(alist):

 for passnum in range(len(alist)-1,0,-1):

for i in range(passnum):

if alist[i]>alist[i+1]:

temp = alist[i]

alist[i] = alist[i+1]

alist[i+1] = temp

Since the outer loop of the bubble sort iterates n times and the inner loop of the outer loop of the bubble sort iterates n times. T(n) is equal to n multiplied by n which is equal to n squared or O(n^2).

**Binary search**

def binarySearch (alist, item):

first=0

last=len(alist)-1

found=False

While first<=last and not found

midpoint=(first+last)/2

If alist[midpoint]==item:

found=True

else:

if item<alist[midpoint]:

last=midpoint-1

else:

first=midpoint+1

return found

Since the while loop of the binary search runs as long as found is false and falso is only true when midpoint(first+last/2) is equal to item passed in the function, T(n) = T(n/2) + 1. You can only half the midpoint up to log2N times. So T(n) would ultimately end up being equal to O(log2N).