**Data Structure and Algorithm**

**Worksheet 2**

Please complete all questions below.

This worksheet forms part of the assessment for this module and you are required to upload your solution on Moodle.

All answers should be typed in your report, in addition to the .java class. You must show all calculations. If you simply give the answer you will not be awarded any marks.

Create a folder called YourName\_StudentNumber\_Assignment02

(e.g. ElaineTynan\_1234567\_Assignment02)

**Part 1 - Analysis**

Question 1:

Using the statement execution times defined for HAL, calculate the running time for the given function.

static int freq(int f[]){ 50+10

int k = 1; int j = 0; 40

while(j < f.length){ 10n+10

if(f[j] \* 2 == j) 50+10+10

k = k \* f[j]; 50 + 10 + 10

j++; 10

}

return k; 50

}

|  |  |  |
| --- | --- | --- |
| Code | Unit Cost | Total Cost |
| static int freq(int f[]){ | 50 + 10 | 60 |
| int k = 1; int j = 0; | 10 + 10 + 10 + 10 | 40 |
| while(j < f.length){ | 10n + 10 | 10n + 10 |
| if(f[j] \* 2 == j) | 50 + 10 + 10 | 70 |
| k = k \* f[j]; | 50 + 10 + 10 | 70 |
| j++; | 10 + 10 | 20 |
| return k; | 50 | 50 |

60 + 40 + 10n + 10 + n \* (50 + 10 + 10 + 50 + 10 + 10 + 10 + 10) + 50 =

160 + 170n

Question 2:

For the following pseudo codes, find the Big-Oh notation

1) **Algorithm** Factorial(a): O(1)

Input: An integer a

Output: The value of a factorial (a!)

factorial <-- 1 O(1)

**for** k=1 **to** a **do O(n)**

factorial <-- factorial \* k N \* O(1) = O(n)

**return** factorial O(1)

Worst Case: O(1) + O(1) + O(n) + O(n) + O(1) = O(n)

Best Case: O(1) + O(1) + O(1) = O(1)

2) **Algorithm** Power(a, b): O(1)

Input: Two integers a and b

Output: The value of a to the power b

power <-- 1 O(1)

**for** k=1 **to** b **do O(n)**

power <-- power \* a O(n)

**return** power O(1)

Worst Case: O(1) + O(1) + O(n) + O(n) + O(1) = O(n)

Best Case: O(1) + O(1) + O(1) = O(1)

3) **Algorithm** LinearSearch(A, n, q): O(1)

Input: An integer array A of size n and a query q that we wish to search the

array for.

Output: The position of q in A or -1 if q is not in A

index <-- 0 O(1)

**while** (index < n) **and** (A[index] <> q) **do O(n)**

index <-- index + 1 O(1)

**if** (index = n) **then**

**return** -1 O(1)

**else**

**return** index O(n)

Worst Case: O(1) + O(n) + O(1) + O(1) + O(n) = O(n)

Best Case: O(1) + O(n) + O(1) = O(n)

**Part 2 – Java Classes**

This next section of the worksheet requires that you write Java code that implements

the algorithms whose pseudo code was given in Part 1.

To verify that you have implmented these algorithms correctly, you are required to

add additional output statements (System.out.println) This means that your main method should

call the corresponding algorithm with the relevant test data.

**Part 3 – Evaluation**

The final part of this worksheet requires that you carry out an experimental evaluation

of each of the algorithms using the experimental framework outlined in the lectures.

For each algorithm, you need to instrument some code so that you can generate timing

information. For non-fixed time algorithms plot the results as a line chart using MS

Excel where the horizontal axis corresponds to n (here n is used to represent the

parameter that affects the performance) and the vertical axis corresponds to time.

**NOTES:**

\* To carry out results smoothing, you should average the performance of the

algorithm over 5 runs.

\* For arrays, use random data over the following values of n (1\*10^6:1\*10^6:10\*10^6)