**Lab sheet 2**

Please complete all the problems described below before next lab.

|  |  |
| --- | --- |
| Calculating Running Times on HAL | |
| **Statement** | **Unit cost (ns)** |
| -, \*, /, %, ^, <, >, ==, >=, <=, !=, = | *10ns* |
| Function invocation | *50ns* |
| Argument passing | *10ns* per argument |
| return | *50ns* |
| if(b) s1; else s2 | the cost of b plus the max cost of s1, s2 |
| for, while loops | *totalCost = cost of initialization of variables +*  *(n+1) \* cost of evaluating guard on loop*  *+*  *n \* cost of executing loop body,*  *where n equals the number of iterations of the loop.* |
| New | *100ns* |
| Calculating array indices | *50ns* |
| Math.random() | *100ns* |

Using the statement execution times defined for HAL (see the table above), calculate the running times for each of the separate code fragments A and B listed below.

// A =========================

int x = 100;

int y = x \* 5 + x \* x - 2;

int z = x + y \* y - 56;

// B ===========================

int k = 0; int s = 0;

while(k < 100){

s = s + (k + 1) \* (k + 1);

}

Using the statement execution times defined for HAL, by calculating the running times for each of the separate code fragments A and B listed below show that the cost of A is *630ns* and that of B is 30 + *30 \* 2 ^ 20ns*. (**Note**: *2 ^ 20* means 2 to the power of 20).

// A ===========================

int k = 1; int N = 2 ^ 20;

while(k < N){ k = k \* 2;}

// B ===========================

int k = 1; int N = 2 ^ 20;

while(k < N){ k = k + 1;}

Using the statement execution times defined for HAL, calculate the running time of the given code fragment.

int f[] = new int[100];

int n = 0;

while(n < 100){

if(n % 2 == 0)

f[n] = 1;

else

f[n] = 0;

n = n + 1;

}

Using the statement execution times defined for HAL, calculate the running time of the given code fragment.

int f[] = new int[100]; //100ns + 10ns

//initialization j cost 10ns

for(int j = 0; j < f.length; j = j + 1) { //loopgard: 101 \* 10ns = 1010ns

f[j] = j\*9; // 70ns \* 100 = 7000ns

}

int sum = 0; //10ns

//10ns

for(int j = 0; j < f.length; j = j + 1){ // 101 \*10ns

if(f[j] % 3 == 0) 20 \* 100 = 2000ns

sum = sum + f[j];

}

Show that function sumN is *O(1)* and sumN1 is *O(n)*. What conclusion can be drawn from this analysis?

**static** **long** sumN(**long** n){

**long** s = n\*(n+1)/2;

**return** s;

}

**static** **long** sumN1(**long** n){

**long** s = 0;

**for**(**int** j=0; j < n; j++) s=s+(j+1);

**return** s;

}