UNIVERSITY OF BIRMINGHAM

M.Sc. CS / AI & CS
Computer Systems
Complexity, Endianness and Intro to OS [Solutions]

Exercise #1: Consider the following fragment of Java code. Although this program does not really do anything, it is instructive to see how we can take actual code and analyze performance. Note: Do not worry about the undeclared variables.

```
int a = 5;
int b = 6;
int c = 10;

for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
      x = i * i;
      y = j * j;
      z = i * j;
   }
}

for (int k = 0; k < n; k++) {
   w = a * k + 45;
   v = b * b;
}

d = 33;
```

a) Formulate an expression to denote the overall execution time of this program considering that each Assignment Operation is treated as a unit of operation.

The number of assignment operations is the sum of four terms. The first term is the constant 3, representing the three assignment statements at the start of the fragment. The second term is $3n^2$, since there are three statements that are performed n^2 times due to the nested iteration. The third term is $2n$, two statements iterated n times. Finally, the fourth term is the constant 1, representing the final assignment statement. This gives us $T(n) = 3 + 3n^2 + 2n + 1 = 3n^2 + 2n + 4$.

b) Identify the dominant term in this expression?

By looking at the exponents, we can easily see that the $3n^2$ term will be dominant.

c) What is the Worst Case Complexity of this program in terms of Big O notation ?

Therefore this fragment of code is $O(n^2)$
Note that all of the other terms as well as the coefficient on the dominant term can be ignored as n grows larger.

UNIVERSITY OF BIRMINGHAM

Exercise #2:

What is the time complexity of following code segments?

a) Code Segment	**Time Complexity**

`int a = 0, b = 0;`
`for (i = 0; i < N; i++) {`
`  a = a + rand();`
`}`
 	
`for (j = 0; j < M; j++) {`
`  b = b + rand();`
`}`
	O(N) + O(M)
	
	= **O(N + M)** time complexity

b) Code Segment	**Time Complexity**

`int a = 0;`
`for (i = 0; i < N; i++) {`
`  for (j = N; j > i; j--) {`
`  a = a + i + j;`
`  }`
`}`
	The above code runs total no of times
	= N + (N – 1) + (N – 2) + … 1 + 0
	= (N * (N + 1)) / 2
	= 1/2 * N^2 + 1/2 * N
	= **O(N^2)**

c) Code Segment	**Time Complexity**

`int i, j, k = 0;`
 	
`for (i = n / 2; i <= n; i++) {`
`  for (j = 2; j <= n; j = j * 2) {`
`  k = k + n / 2;`
`  }`
`}`
	If you notice, j keeps doubling till it is less than or equal to n. Number of times, we can double a number till it is less than n would be log(n).
	Let's take the examples here.
	for n = 16, j = 2, 4, 8, 16
	for n = 32, j = 2, 4, 8, 16, 32
	So, j would run for O(log n) steps.
	i runs for n/2 steps.
	So, total steps = O(n/2 * log (n))
	
	= **O(n log n)**

d) Code Segment	**Time Complexity**

`int numberFound = 0`
`for i = 0..(n-1)/2`
`  for j = 0..(n-1)/2`
`  if BinarySearch(A[i,j])`
`  numberFound = numberFound + 1`
	Complexity of each loop is O(n)
	
	Complexity of Binary Search is O(log n)
	
	Overall complexity is: **O(n^2 log n)**

UNIVERSITY OF BIRMINGHAM

Exercise #3: A program you have developed has a fixed startup time and a main algorithm which you know is $O(n^3)$. You have tested the program with 1 data item, and it takes 10 seconds to execute. With 100 data items it takes 15 seconds to run. Estimate how long (in seconds) this algorithm will take for 400 items?

With 1 item it takes 10 seconds. This means that it has a fixed startup cost of approximately 10 seconds (the time to process 1 item is negligible).
With n=100 it takes 15 seconds: 10 seconds startup time + 5 seconds to process the 100 items.
The main algorithm is $O(n^3)$.
Therefore, if we have 4 times as much data, we can estimate that it will take $4^3 = 64$ times as long to execute that part of the program + the fixed startup time.
[If we double the amount of data then we would increase the time 2^3 times = 8 times. If we doubled it again, it will be 4^3 times = 64 times and so on]
Therefore, it will take 10 + 64*5 = 10 + 320 = 330 seconds

Exercise #4: A sorting method with "Big-O" complexity O(n log n) spends exactly 1 millisecond to sort 1,000 data items. Assuming that time T(n) of sorting n items is directly proportional to n log n, that is, T(n) = c n log n, derive a formula for T (n), given the time T (N) for sorting N items, and estimate how long this method will sort 1,000,000 items.

Because the processing time is T(N) = c N log N, the constant factor c = T(N) /(N log N)
We can replace the value of c in T(n) = c n log n, and get T(n) = (T(N) * n log n) /(N log N).
Ratio of the logarithms of the same base is independent of the base, hence, any appropriate base can be used in the above formula (lets say base of 10).

Therefore, for n = 1,000,000 items, the time is:

$$T(1000000) = (T(1000) * 1000000 \log_{10} 1000000) / (1000 \log_{10} 1000)$$
$$= (1 * 1000000 * 6) / 1000 * 3)$$
$$= 2000 \text{ ms}$$

Exercise #5: Show how the following values would be stored in a byte-addressable machines with 32-bit words, using little endian and then big endian format. Assume that each value starts at address 1016. Draw a diagram of memory for each, placing the appropriate values in the correct (and labeled) memory locations.

- 0x456789A1
- 0x0000058A
- 0x14148888

For 0x456789A1

Memory Address	**1016**	**1017**	**1018**	**1019**
Little Endian	A1	89	67	45
Big Endian	45	67	89	A1

For 0x0000058A

Memory Address	**1016**	**1017**	**1018**	**1019**
Little Endian	8A	05	00	00
Big Endian	00	00	05	8A

For 0x14148888

Memory Address	**1016**	**1017**	**1018**	**1019**
Little Endian	88	88	14	14
Big Endian	14	14	88	88

Exercise #6: We have stressed the need for an operating system to make efficient use of the computing hardware. When is it appropriate for the operating system to forsake this principle and to “waste” resources? Why is such a system not really wasteful?

Single-user systems should maximize use of the system for the user. A GUI might “waste” CPU cycles, but it optimizes the user’s interaction with the system.

UNIVERSITY OF BIRMINGHAM

Exercise #7: How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security) system?

The distinction between kernel mode and user mode provides a rudimentary form of protection in the following manner. Certain instructions could be executed only when the CPU is in kernel mode. Similarly, hardware devices could be accessed only when the program is executing in kernel mode.

Control over when interrupts could be enabled or disabled is also possible only when the CPU is in kernel mode. Consequently, the CPU has very limited capability when executing in user mode, there by enforcing protection of critical resources.

Exercise #8: Which of the following instructions should be privileged?

a) Set value of timer.

b) Read the clock.

c) Clear memory.

d) Issue a trap instruction.

e) Turn off interrupts.

f) Modify entries in device-status table.

g) Switch from user to kernel mode.

h) Access I/O device.

The following operations need to be privileged:

a) Set value of timer

c) Clear memory

e) Turn off interrupts

f) Modify entries in device-status table

h) Access I/O device.

The rest can be performed in user mode.