Mahindra École Centrale

Second Mid-Term Examination

CS101: Introduction to Computer Science

Time: 1 hour 30 minutes March 25,2015 **Maximum Marks: 100**

Problem I (*Marks:* 30)

This first problem is a list of **Multiple Choice Questions**. Each question (from Q.I.1 to Q.I.15) carries 2 marks, and has only one correct answer.

Q.I.1) The worst case occurs in linear search through a sorted list when _____

a) item is somewhere in the middle of the array
b) item is not in the array at all
c) item is the last element in the array
d) item is the last element in the array or item is not there at all

Q.I.2) Best known time complexities of search algorithms for *sorted* and *unsorted* lists are (a pair) _____, where n denotes the problem size

a) (O(n), O(log(n)))
b) (O(log(n)), O(n))
c) (O(log(n)), O(log(n)))
d) (O(n), O(n^2))

Q.I.3) Best known time complexity of sorting an unsorted list is O(nlog_2(n)). Given an unsorted list of length 128, and tasked with making k number of searches through the list. At what value of k onwards, would you prefer to first sort the list once and then use binary search every time, instead of repeatedly searching for items in the original unsorted list?

a) 1 **b)** 7 **c)** 8 **d)** 20

Q.I.4) The complexity of bubble sort algorithm is _____

a) O(n) **b)** O(logn) **c)** O(n^2) **d)** O(n logn)

Q.I.5) Which of the following sorting algorithm is of divide and conquer type?

a) Bubble sort **b)** Insertion sort **c)** Merge sort **d)** Selection sort

Q.I.6) What is the output when we execute the below code?

```
1  Matrix = [[1,2,3],[4,5,6],[7,8,9]]
2  c=0
3  for i in range(3):
4      for j in range(3):
5          if(i==j):
6              c=c+Matrix[i][j]
7  print "Result is", c
```

a) 12 **b)** 14 **c)** 18 **d)** 15

Q.I.7) Which is not a built-in module in Python?

a) math **b)** sys **c)** os **d)** file

Q.I.8) Which one of the module import methods is invalid in Python?

a) from math import sqrt
b) from math.sqrt import
c) from math import sqrt as square_root
d) from math import sqrt as sqrt

Q.I.9) In Python, which function can be used to find the functions and attributes of a module?

a) list() **b)** dir() **c)** display() **d)** ls()

Q.I.10) Which file designates a directory as a package in Python?

a) ___main___.py **b)** ___init___.py **c)** ___dir___.py **d)** ___pkg___.py

Q.I.11) What is the value of this expression which uses **list comprehension**?

```
1 s = sum([ i for i in range(20)  if (i%2 == 0) and (i%5 == 0) ])
```

a) `s = 10`
b) `s = [2, 4, 5, 6, 8, 10, 12, 15, 16, 18]`
c) `s = 30`
d) `s = [10]`

Q.I.12) What sequence of Python operations will produce the list `l = [2, 6, 8]` ?

a)

```
1 l = []
2 l.append(2)
3 l.append(6)
4 l.append(8)
```

b)

```
1 l = [8]
2 l.append(2)
3 l = [2] + l * 2
4 l.insert(1, 6)
5 l.pop()
```

c)

```
1 l = [8]
2 l += [6]
3 l += [2]
```

d)

```
1 l = list({ 2, 6, 8 })
2 l = l
3 l = list(l)
```

Q.I.13) Which of the following set exists?

a) `{ -1, 1, 0, -1, 1 }`
b) `{ 2+3j, 0, 2-3j }`
c) `{ -1, [0], 1 }`
d) `{ 4, {-1, 1}, [4+6j] }`

Q.I.14) To create a **dictionary** that maps some integers to their cube, which of the following expression is good?

a) `d = { i**3 : i for i in range(12) }`
b) `d = { i : i**3 for i in range(12) }`
c) `d = { (i, i**3) for i in range(12) }`
d) `d = [ i : i**3 for i in range(12) ]`

Q.I.15) If we have a (nested) **dictionary** representing a database of students (mapping roll numbers to a dictionary for each student), and you want to print all their email address, what can you try?

a)

```
for roll in students:
    print "Student of roll number", roll, "has this email address:", ↩
        students['roll']['email']
```

b)

```
for guy in students.values():
    print "Student of roll number", guy['roll'], "has this email address:",↩
         guy['email']
```

c)

```
for roll, guy in students.items():
    print "Student of roll number", roll, "has this email address:", guy['↩
        roll']['email']
```

d)

```
for roll in students.values():
    print "Student of roll number", roll, "has this email address:", ↩
        students[roll]['email']
```

Problem II (*Marks:* 20)

We look at the task of exponentiation of any real quantity 'a' to any positive integer 'b', i.e. a^b. To keep matters simple, we constrain 'b' to be a power of 2 only, i.e. b = 2^k for some k = 1, 2, ... The code for this exponentiation is provided below:

```
# Start of program
def exp3(a, b):
    print 'a= ',a,' b= ', b
    if (b == 1):
        return a
    else:
        p = a * a
        return exp3(p, b/2)
# end of function

a = input("Enter the base value 'a' ")
b = input("Enter the exponent 'b' - should be a power of 2 ")
result = exp3(a, b)
print result
```

Now, you are to do the following:

Q.II.1) Derive the time complexity of the above computation, as O(f(b)), where f(.) is some function that you need to derive clearly and logically, e.g. could be of form b^2, 2^b, etc.. (*Marks:* **5**)

Q.II.2) The above so far considers 'a' to be any real number. Replace both 'a' and 'p' to be matrices A and P of size (n × n), and P = A × A. Accordingly, replace the statement p = a * a with a call to a function to multiply two square matrices
P = sqr_mat_mul (A, A, n)
and correspondingly modify the rest of the above function.
Write the function for sqr_mat_mul (A, A, n). (*Marks:* **5**)

Q.II.3) Evaluate the time complexity of the matrix multiplication O(g(n)), where g(.) is some function that you need to derive logically and unambiguously. (*Marks:* **5**)

Q.II.4) Finally evaluate the time complexity of the complete program to raise a square matrix A of order (n x n) to the exponent b where b is as defined before. This time complexity will be of form O(h(n, b)) where h(. , .) is a function of both n and b. (*Marks:* **5**)

Problem III (*Marks:* 23)

This problem is about lists of numbers, and some mathematical operations that we can compute on such lists.

Q.III.1) Write a simple function `max` and `min` that computes the **maximum** and **minimum** of a list `l` of numbers.

Do not use the *built-in* function `max` and `min`, **you need to program them yourself!** (*Marks:* **3**)

You will be cautious about the case of an empty list `l = []`, for which min and max are not defined (you can raise an exception with the command `raise ValueError("min() arg is an empty sequence")`)

Q.III.2) Similarly, write a small function `sum` that computes the **sum** of a list `l` of numbers.

Mathematically, an empty sum (ie. `sum([])`) is *defined* as 0. *Be sure to include this case.*

Again, **do not use** the built-in function `sum`, you need to program it yourself! (*Marks:* **2**)

Q.III.3) Now, write a function called `average` that computes (and *returns*) the arithmetic average of the numbers in the list `l`: `average`$([x_0, \dots, x_{n-1}]) = \frac{1}{n}\left(\sum_{k=0}^{n-1} x_k\right)$ for a non-empty list of size n `= len(l)`.(*Marks:* **3**)

Be cautious about the empty list!

Examples: `average([0, 1]) = 0.5`, `average([1, 2, 3, 4, 5]) = 3.0`, and `average([-1, 1, 3]) = 1.0` for instance.

Q.III.4) We are now interested about computing the **weighted average** of these numbers. Assume you have a list of numbers `l` $= [x_0, \dots, x_{n-1}]$, and a list of **weights** `w` $= [w_0, \dots, w_{n-1}]$, of the same sizes.

Write a function `weighted_average` that will accept two arguments `l` and `w`, and will compute (and *return*) the weighted average: `weighted_average`$([x_0, \dots, x_{n-1}], [w_0, \dots, w_{n-1}]) = \frac{\left(\sum_{k=0}^{n-1} w_k x_k\right)}{\sum_{k=0}^{n-1} w_k}$ for two non-empty lists `l, w` of the *same size* n.

You can chose to raise an exception if they have a different size (with something like `raise ValueError("weighted_average() excepts two lists of the same size.")`). (*Marks:* **5**)

Hint: the usual average is like the weighted average with equal weights $w_0 = \cdots = w_{n-1} = 1$.

Q.III.5) What is the **time complexity** (ie. *approximate* number of elementary operations) of your `average` function? (*Marks:* **5**)

Similarly, what is the time complexity of `weighted_average`, as a function of n, common size of the two lists `l` and `w`?

Both time complexities will be functions of n, the size of the argument list `l`. These approximated complexities should be something like $O(\log(n))$, $O(n)$, $O(n\log(n))$, $O(n^2)$ or $O(n^3)$ for example.

Q.III.6) Now write two functions `geom_mean` and `harmo_mean` that accepts a list of **positive numbers** (ie. $x_0 > 0, \dots, x_{n-1} > 0$), and computes (and *returns*) respectively the geometrical mean and the harmonic mean. (*Marks:* **5**)

Definitions:

- `geom_mean`$(x) = \left(\prod_{k=0}^{n-1} x_k\right)^{1/n}$ for a non-empty list `x` of size n `= len(x)`,
- `harmo_mean`$(x) = \frac{n}{\left(\sum_{k=0}^{n-1} 1/x_k\right)}$ for a non-empty list `x` of size n `= len(x)`.

Hint: No need to check that the numbers are positive or not.

Problem IV (*Marks:* **27**)

Here we investigate further the problem of sorting a list of numbers using the "Merge Sort" Algorithm. The code structure for merge sort is presented below, on similar lines to the code that you have already seen. You are to do the following:

```
1
2     def merge(left, right):
3       # Assumes 'left' and 'right' are sorted lists.
4       #Puts them together in a new sorted list 'result'.
5       result = [0.0]
6       i = 0
7       j = 0
8       while ((i < len(left)) or (j < len(right))):
9       # Going over a condition-based dual loop over both
10      # 'left' and 'right'
11        if (left[i] < right[j]):
12            result.append(left[i])
13            i += 1
14        else
15            result.append(right[j])
16            j += 1
17        # end if-else construction
18        # end while loop
19        # It is unlikely that both 'right' and 'left' have reached
20        # their max values simultaneously. Hence, below code
21        # completes the remaining values.
22        while (i < len(left)):
23            result.append(left[i])
24            i += 1
25        while (j < len(right)):
26            result.append(right[j])
27            j += 1
28        return result
29    # end of function
30
31    def merge_sort(s):
32        if (len(s) <= 2):
33            return s[:]
34        else:
35            mid = len(s)/2.0
36            right = merge_sort(s[0:mid])
37            left = merge_sort(s[mid:])
38            combine = merge(left, right)
39            return combine
40        # end of if-else condition
41    # end of function
42
43    s = [9, 3, 7, 0, 1, 8, 5, 4]
44    print s
45    sorted_s = merge_sort(s)
46    print "In call, from returned ", sorted_s
```

Q.IV.1) There are four semantic errors and two syntax errors injected into this code. You are to spot these errors.

Recall that a syntax error will be captured by an interpreter and the code will not run, but a semantic error will not get identified by the interpreter as that is intrinsic to the manner of mapping of the algorithm into the code, and hence the code will continue to run but give wrong results.

Each semantic error carries 4 marks, and each syntax error two. (*Marks:* **20**)

Q.IV.2) See the illustration below. This shows the sequence of steps in the merge sort execution.

You will find the split into a "Division Zone" and a "Merge Zone", the sizes of lists for an initial size of eight as execution proceeds, and a dashed curve at the left (hypothetical) to show the initial divisions till the first "return" step is encountered. (if unable to comprehend what this means, try to mentally dissect the function "merge_sort").

Now you are to provide a similar illustration on your answer sheet, with actual numbers – as printed in the list provided in the code – representing the evolution of the merge sort process from start to finish.

Importantly, you are to provide a dashed curve encapsulating the initial few division boxes before the first "return" is encountered anywhere.

Hint: Look closely at the lines of code else you will be misled. **(*Marks:* 7)**

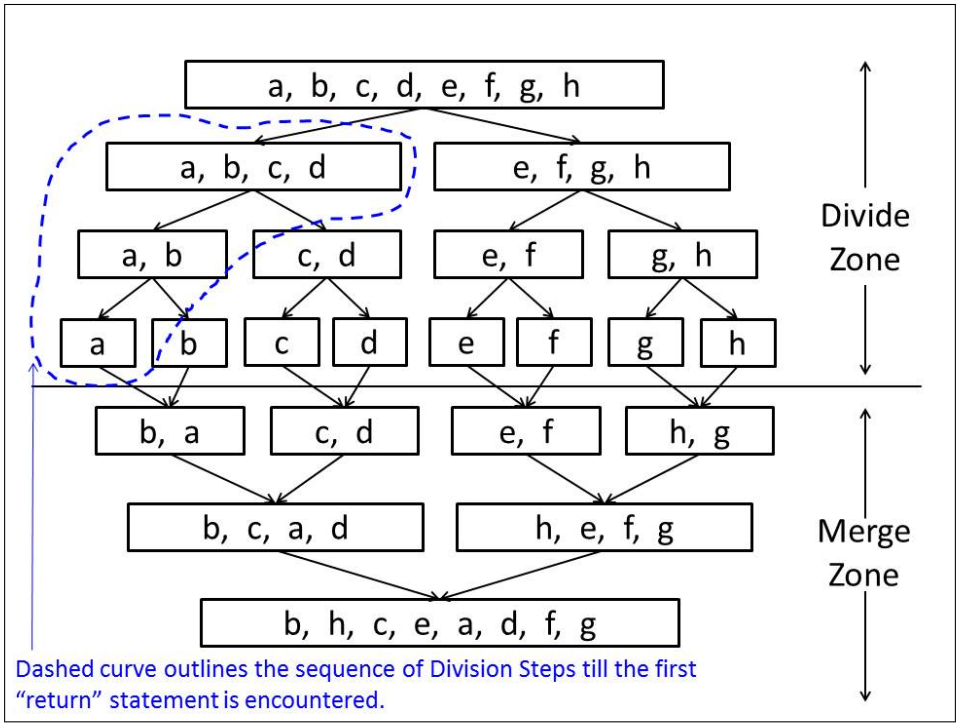

Figure 1: Merge sort illustration.