The University of the South Pacific

School of Computing, information and Mathematical Sciences

CS214: Design & Analysis of Algorithms

Test I Semester 2, 2018

Time Allowed: 50 mins
Total Marks: 10 (10% of final grade)

Name: ________________________ **Student ID:** ________________________

Campus: ________________________

NOTE:

This test covers the two learning outcomes:

1. Evaluate the efficiency of algorithms (7%)
2. Assess the suitability of different algorithms for solving a given problem (3%)

INSTRUCTIONS:

- All questions are compulsory.
- All answer must be written in the spaces given in this booklet.
- You are not allowed to use the course material or internet search during the exam.

1. Suppose USP library uses an efficient online system to search for the books. The book names are stored in an appropriate data structure, along with its authors, editions, publisher etc. Library users search for a book by providing the title. The system picks the key words and displays the result. For example "algorithm" word may display many titles such as "data structures & algorithms", "complexity of algorithms", "writing algorithm in C++" etc. What is the best data structure for this scenario. Justify your answer. (1+2 marks)

(10 mins)

I would choose Hash-table/2D-linked list/heap/priority queue ds. (retrieval has higher priority than insertion)

Why?

Heap is efficient and has priority structure that means:

- heap is a complete tree (All levels are completely filled except possibly the last level). Vertical -> [A-Z]

  This would result in $\log_k n$ in average case…

- The value of each node is greater (or less) than or equal to the value of its parent.

  Use: higher priority i.e. commonly used words would be placed on top.

  How to get the priority?

  Through indexing by keep tracking of usage of words. The higher usage would result in higher priority.

2. Some algorithms can be written in iterative as well as in recursive forms. Suppose algorithm **X** has been written using both styles, however, it was found that iterative approach is somewhat slower than the recursive approach. The number of operations required by both approaches for different array sizes are listed below:

Size	Iterative	Recursive
1	5	2
2	10	4
3	15	6
4	20	8
…	…	…

Do you agree with the claim of the testers? Please explain. (1+2 marks)

(10 mins)

Yes, no – 1 mark.

Actually both have linear time complexity. It won't matter which one to use.

~~Iterative are generally easy to use so I will prefer iterative even though it is slightly slower but computationally the difference is negligible.~~

3. Find the every-case time complexity of following code:

```
D = 2
for i = 1 to n do
   for j = i to n do
      for k = j + 1 to n do
           D = D * 3
```

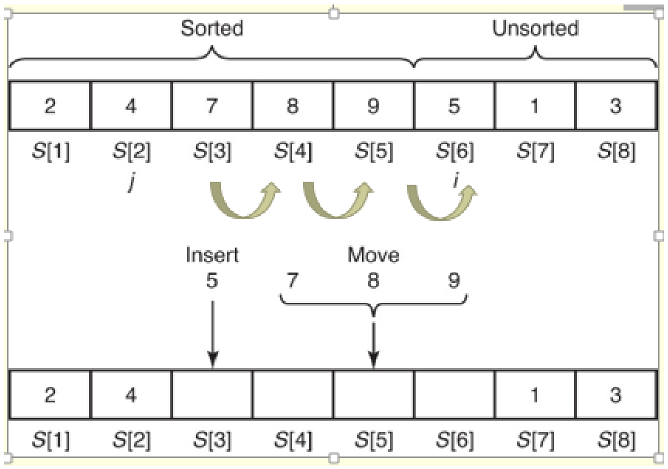
Show your working. (1 + 1 marks)

```
(10 mins)

Go from bottom to up...

Loop j runs n times and for each time k runs:


I=1; n-1 + n-2 + … 1 times = n*(n+1)/2 = n^2/2 + n/2 approx. n^2

I=2; n-2 + n-3 + … 1 times

I=3; n-3 + n-4+ … 1 times

…

I=n; n-n = 1 time

I runs n times. So in total

N*n^2 = n^3.
```

4. What is the Big O order of the average time complexity for the *insertion sort*? The algorithm and an example illustrating what *insertion sort* does when $i = 6$ are shown below:

(2 marks + 1 bonus)

```
/*Function to sort array using insertion sort*/
void sort(int arr[])
{
    int n = arr.length;
    for (int i=1; i<n; ++i)
    {
        int key = arr[i];
        int j = i-1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while (j>=0 && arr[j] > key)
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = key;
    }
} //Ref: https://www.geeksforgeeks.org/insertion-sort
```



(20 mins)

$$E(X) = pass * P(e1) + pass * P(e2) + \cdots + pass * P(eN)$$

To move 5 it can either go to s[5] or s[4] …. Or s[1]

So what is the probability?

It would be 1/k here k = location of 5 which is 6. i.e. probability is 1/6.

So only for element 5 it will have:

$E(s[6]) = \frac{1}{k} * 1 + \frac{1}{k} * 2 + \cdots + \frac{1}{k} * k$ //1 pass or 2 pass or up to k passes with same probability.

So overall

$E(s[1]) = \frac{1}{1} * 1 \qquad = \frac{1}{1} * 1 \quad = \frac{2}{2}$

$E(s[2]) = \frac{1}{2} * 1 + \frac{1}{2} * 2 \qquad = \frac{1}{2} * \frac{2*3}{2} = \frac{3}{2}$

$E(s[3]) = \frac{1}{3} * 1 + \frac{1}{3} * 2 + \frac{1}{3} * 3 \qquad = \frac{1}{3} * \frac{3*4}{2} = \frac{4}{2}$

…

$E(s[n]) = \frac{1}{n} * 1 + \frac{1}{n} * 2 + \cdots + \frac{1}{n} * n = \frac{1}{n} * n \frac{(n+1)}{2} = \frac{n+1}{2} = \frac{n}{2}$ //actually last one will be n-1.

$\therefore E(s) = \frac{2}{2} + \frac{3}{2} + \frac{4}{2} + \cdots + \frac{n}{2} = \frac{1}{2} * \frac{n(n+1)}{2} - 1 \cong n^2$ hence Big O is $O(n^2)$

Extra sheet:

END