You are not allowed to tear pages from booklets - this is against univ. rules.

COMP 6651: Algorithm Design Techniques
Winter 2016 Mid-term Examination

Time: 1 hour 30 minutes

Student name: Xintian Wang **Student id:** 40016301

Notes:

1. This exam has 4 questions, each worth 5 marks, and has 8 pages. Read every question very carefully before attempting to solve it.
2. Answers must be short and precise yet complete. Verbose explanations may reveal flaws in your logic, which would incur penalties.
3. You do not need to give the pseudocode of any algorithm that we have studied in class, simply call it as a function/subroutine with the appropriate parameters.

5
4
3
5
17

You may or may not wish to make use of the following facts:

(i) $\Sigma_{k=1}^{n} k = \frac{n(n+1)}{2}$

(ii) $\Sigma_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$.

(iii) $\log_a a^i = i$

(iv) $\log_b a = \frac{\log_c a}{\log_c b}$

(v) **Master Theorem**: Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be defined on the non-negative integers by the recurrence:

$$T(n) = aT(n/b) + f(n)$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$, Then $T(n)$ can be bounded asymptotically as follows:

(a) If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$.

(b) If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$.

(c) If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n then $T(n) = \Theta(f(n))$.

(vi) **Sorting and Selection Algorithms and their Complexities**

Algorithm	Worst-case time	Average-case time	Extra Space
Linear search	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$
Binary search	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
Quicksort	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(\log n)$
Mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$
Heap sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(1)$
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$
Counting sort*	$\Theta(n + k)$	$\Theta(n + k)$	$\Theta(k)$
Minimum/Maximum	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$
Median	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$
k-th largest element	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$

* assuming that elements are in the range $[1, \ldots, k]$.

1. Solve the recurrence $T(n) = 16T(n/4) + n$.

(a) (2 marks) Use the master theorem to generate a guess. Show all your work, state what are $a, b, \log_b a, n^{\log_b a}, f(n)$, which case you are using, ϵ if applicable, and the solution in proper asymptotic notation.

Use master theorem $a=16$ $b=4$ $\log_b a = 2$ $f(n)=n$

$n^{\log_b a} = n^2$

use case 1: $f(n) = O(n^{2-\varepsilon})$ when $0 < \varepsilon \leq 1$

$\therefore T(n) = \Theta(n^2)$

2

(b) (3 marks) Prove an upper bound on $T(n)$ using the substitution method. Your answer must be in O-notation. You do not need to prove a lower bound, and you may ignore floors and ceilings.

Suppose $T(n) \leq cn^2 - n$ $c > 0$

substitution $T(n) = 16T(\frac{n}{4}) + n \leq 16(\frac{cn^2}{16} - \frac{n}{4}) + n$

$= cn^2 - 3n$

$= (cn^2 - n) - 2n < cn^2 - n$

$\therefore T(n) = O(n^2)$

3

2. You are given an unsorted array A of n elements containing integers in the range 1 to m. You want to answer queries of the sort "How many array elements are in the range $[a, b]$?" where a and b are integers. Design an $O(max(m,n))$ algorithm to preprocess the array so that any such query can be answered in $O(1)$ time.

(a) (1 mark) Give a brief description in English of your idea.

~~Use counting sort~~ ~~let C[1]~~

let C[1..m] to store the number the numbers which equals to m, then accumulate it, then C[k] store the number of numbers which ~~[illegible]~~ $\leq k$, then $r[a,b] = C[b] - C[a-1]$

1

(b) (2 marks) Give the pseudocode for the pre-processing algorithm. Explain briefly why it takes $O(max(m, n))$ time.

```
pre-prass(A, m, n)
  for i=1 to m
    C[i] = 0
  for j=1 to n
    C[A[j]] = C[A[j]] + 1
  for i=2 to m
    C[i] = C[i-1] + C[i]
  return C
```

~~[a,b] = C[b] - C[a-1]~~

$T(n) = \Theta(m) + \Theta(n) + \Theta(m)$

$\leq cm + cn + cm$

$\leq d\,max(n, m)$

$\therefore T(n) = O(max(m,n))$

1

(c) (2 marks) Give the pseudocode for the query-response algorithm. Explain briefly why it takes $O(1)$ time.

```
Query(A, a, b)
  C = prepross(A, m, n)
  return C[b] - C[a-1]
```

~~the time is O(1)~~ the second line use time $\Theta(1)$

2

3. Given a set of points $a_1, a_2, \ldots, a_n$ on a line, with $a_1 \leq a_2 \leq \ldots \leq a_n$, you want to find a minimum sized set of unit intervals $\{[p_i, p_i + 1] \mid 1 \leq i \leq k\}$ that contain the points. For example, suppose the given set of points is $\{2.5, 3.3, 5, 5.9, 6.3\}$, then an optimal solution is the set of intervals $\{[2.5, 3.5], [4, 5], [5.5, 6.5]\}$, as it has only three intervals, and there is no solution covering the set of points that uses only two intervals. There are many other optimal solutions. However $\{[1.5, 2.5], [3, 4], [5, 6], [6, 7]\}$ is a non-optimal solution as it uses four intervals.

(a) (1 mark) Describe in words a greedy heuristic to solve the problem.

~~For any input $A = a_1, \ldots, a_n$, choose~~

For any input $A_j = \{a_j, a_{j+1}, \ldots, a_n\}$, always choose the $[a_j, a_j + 1]$ as the first interval, then find the $a_k > a_j + 1$,

(b) (2 marks) State precisely the property of greedy choice *specific to your heuristic*. There is no need to give a proof, just state the property.

recursive do next for $A_k = \{a_k \ldots a_n\}$

Suppose the optimal solution is $P = (p_1, p_2 \ldots p_k)$

that the unit is $\{[p_1, p_1+1], [p_2, p_2+1] \ldots [p_k, p_k+1]\}$

Exists an optimal solution that contains the greedy choice.

the greedy choice is $[A_j, A_{j+1}]$

(c) (2 marks) Give the pseudocode for your greedy algorithm.

```
Minimum_set_intervals (A)
    n = A.length
    let p[1..n] a new array
    p[1] = a_1
    s = 1
    k = 1
    while a_n > p[k] + 1
        i = s + 1
        while i ≤ n
            if a[i] ≤ p[k] + 1
                i = i + 1
        if i ≤ n  s = i
                  k = k + 1
return P, k        p_k = a_i
```

~~for i = 1 to k~~

4. We say that a string is a *palindrome* if it reads the same backwards and forwards. Some examples of palindromes are the strings *a*, *aabbaa*, *racecar* and *malayalam*. We want to design an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input *character*, your algorithm should return *carac*. Assume you are given an array $A[1..n]$. Give a recursive formulation for $c[i, j]$, which is defined to be the length of the longest palindrome that is a subsequence of the sub-array $A[i..j]$. Assume that $c[i, j] = -\infty$ if $i > j$.

(a) (1 mark) For $1 \leq i \leq n$, what is $c[i, i]$?

$c[i,i] = 1$

(b) (1 mark) For $1 \leq i \leq n-1$, what is $c[i, i+1]$?

$$c[i,i+1] = \begin{cases} 1 & a_i \neq a_{i+1} \\ 2 & a_i = a_{i+1} \end{cases}$$

(c) ((2 marks) For $1 \leq i < j \leq n$, what is $c[i, j]$?

$$c[i,j] = \begin{cases} c[i+1, j-1] + 2 & a_j = a_i \\ \max(c[i, j-1], c[i+1, j]) & a_i \neq a_j \end{cases}$$

(d) (1 mark, only if the previous part is correct) State the time complexity of implementing your recursive formulation using a dynamic programming bottom-up method, and give a brief explanation. There is no need to give the pseudocode.

Using bottom-up method, we have a table of n^2 to fill, and each entry need $O(1)$ time. So the time complexity is $O(n^2)$