INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR		
		Stamp / Signature of the Invigilator

EXAMINATION (End Semester)										**SEMESTER (Spring)**			
Roll Number										**Section**		**Name**	
Subject Number	C	S	2	1	0	0	3	**Subject Name**		*Algorithms – I*			
Department / Center of the Student											**Additional sheets**		

Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession, even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items or any other papers (including question papers) is not permitted.
6. Write on both sides of the answer script and do not tear off any page. **Use last page(s) of the answer script for rough work.** Report to the invigilator if the answer script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the examination hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer script to the invigilator. **In any case, you are not allowed to take away the answer script with you.** After the completion of the examination, do not leave the seat until the invigilators collect all the answer scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging information with others or any such attempt will be treated as **'unfair means'**. Do not adopt unfair means and do not indulge in unseemly behavior.

Violation of any of the above instructions may lead to severe punishment.

Signature of the Student

To be filled in by the examiner

Question Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Marks Obtained											

Marks obtained (in words)	**Signature of the Examiner**	**Signature of the Scrutineer**

CS21003 Algorithms–I, Spring 2017–2018

End-Semester Test

24–April–2018 V4, NR121/122/221/322/422, 02:00pm–05:00pm Maximum marks: 60

Write your answers in the question paper itself. Be brief and precise. Answer all questions.
If you use any algorithm/result/formula covered in the class, just mention it, do not elaborate.

1. In a *ternary search tree* T, each node contains two keys key_1 and key_2, and three child pointers $left$, mid, and $right$. Let v be any node in T, k_1 a key stored in any node in the subtree rooted at $left(v)$, k_2 a key stored in any node in the subtree rooted at $mid(v)$, and k_3 a key stored in any node in the subtree rooted at $right(v)$. Then, we must have $k_1 < key_1(v) < k_2 < key_2(v) < k_3$. A ternary search tree is called *admissible* if the three subtrees at any node of the tree differ in height by at most one.

(a) Let $N(h)$ denote the minimum number of nodes in an admissible ternary search tree of height h. Derive a recurrence relation for $N(h)$. Also, supply the required initial condition(s). **(4+1)**

Solution Let T be an admissible ternary search tree of height h and with the minimum possible number $N(h)$ of nodes. Since $ht(T) = h$, at least one of the three subtrees of T must have height $h-1$. In order to minimize the number of nodes in T, the other two subtrees should have height $h-2$ each (otherwise T is not admissible). This gives the following recurrence relation.

$$N(h) = 1 + N(h-1) + 2N(h-2) \text{ for } h \geqslant 2.$$

The initial conditions are now stated.

$$N(0) = 1,$$
$$N(1) = 2.$$

(b) Let h be the height of an admissible ternary search tree with n nodes. Deduce that $h \leqslant \log_2 n$. (This result shows that admissible ternary search trees are height-balanced.) **(5)**

Solution We have $n \geqslant N(h)$, so it suffices to derive a lower bound on $N(h)$. We establish by induction that $N(h) \geqslant 2^h$ for all $h \geqslant 0$. For $h = 0, 1$, this claim holds. So suppose that $h \geqslant 2$. But then, by induction, $N(h) = 1 + N(h-1) + 2N(h-2) \geqslant 1 + 2^{h-1} + 2 \times 2^{h-2} = 1 + 2^h \geqslant 2^h$. It follows that $n \geqslant 2^h$, that is, $h \leqslant \log_2 n$.

2. The *Manhattan distance* between two points (h,k) and (h',k') in the Euclidean plane is defined as $|h-h'|+|k-k'|$. You are given n points $P_i=(x_i,y_i)$, $i=0,1,2,\ldots,n-1$, randomly chosen in the unit square (that is, the 1×1 square with corners at $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$). The problem is to find, given a query point $Q=(x,y)$ in the unit square, a point P_i closest to Q with respect to the Manhattan distance. Your task is to organize the given points P_i in a data structure such that each query can be answered in expected constant time. Propose a suitable data structure for storing the points P_i. Also state the algorithm how each query is answered. Argue that each query can be processed in expected constant time. **(4+4+2)**

Solution Let $m=\lceil\sqrt{n}\,\rceil$. Store the n points P_i in a two-dimensional hash table T of size $m\times m$. The unit square is broken into an $m\times m$ mesh, and each entry of T stands for an $\frac{1}{m}\times\frac{1}{m}$ square in the mesh. Use a hash function $h(x,y)=(i,j)$, where x,y are real numbers in $[0,1]$, and $i,j\in\{0,1,2,\ldots,m-1\}$. Multiple points P_i hashing to the same cell in the two-dimensional table T are stored by chaining.

If the input points P_i are uniformly randomly distributed in the unit square, a natural choice for h is to take $i=\min(\lfloor mx\rfloor,m-1)$ and $j=\min(\lfloor my\rfloor,m-1)$. If this assumption on the distribution of P_i does not hold, then h should introduce the requisite randomness, that is, for all $x,y\in[0,1]$, the hash value $(i,j)=h(x,y)$ should be a uniformly random element of $\{0,1,2,\ldots,m-1\}^2$.

Now, let $Q=(x,y)$ be a query point. We first compute $(i,j)=h(Q)$. For all points P_i stored in the nine cells $(i+\delta,j+\varepsilon)$ with $\delta,\varepsilon\in\{-1,0,1\}$, the Manhattan distances between P_i and Q are computed. The point P_i found closest to Q is reported.

If none of the nine cells contains any P_i, we expand the search space by incrementally taking $\delta,\varepsilon=2,3,\ldots$ until a point P_i closest to Q is located.

If the points P_i are uniformly randomly chosen in the unit square, each $\frac{1}{m}\times\frac{1}{m}$ cell is expected to contain about one of the points P_i (notice that $n\approx m^2$). Consequently, the search for a closest point to Q is expected to terminate after $O(1)$ possibilities of δ,ε are tried, that is, we expect to look into at most $O(1)$ cells in the hash table T until the search succeeds.

3. Let $G = (V, E)$ be a connected undirected graph with n vertices and (exactly) n edges.

(a) What are the minimum and the maximum numbers of spanning trees that G can have? Justify. **(3)**

Solution The given conditions imply that G contains a unique cycle C. Removing any edge from C gives a spanning tree of G, so the number of spanning trees in G is equal to the length of C. This length is in the range $3, 4, 5, \ldots, n$, so the desired minimum and maximum numbers are 3 and n.

(b) Let each edge e of G carry a positive cost (or weight) $c(e)$. Propose an $O(n)$-time algorithm to compute an MST (a minimum spanning tree) of G. Comment on what data structures you used in your algorithm. **(5+2)**

Solution Let us run a DFS traversal from any source. The cycle in G shows up as a backward/forward edge (v, u), where u is an ancestor of v. The tree edges from u to v along with the backward edge (v, u) is the cycle in G. Remove an edge of maximum cost from this cycle to get an MST of G. The cycle lemma proves the correctness of this algorithm. The DFS traversal runs in $O(|V| + |E|)$, that is, $O(n)$ time. Finding the cycle and removing the largest-cost edge in the cycle can also be done in $O(n)$ time.

In order to let the DFS traversal finish in linear time, we can use the adjacency-list representation of G. The DFS tree can be stored in the parent-pointer representation. At each non-root node u with parent p, we may store (along with the parent pointer) the cost of the edge (p, u). This avoids looking at the adjacency lists during the cycle-finding phase. However, since the total size of all the adjacency lists is $O(n)$ in this case, this extra storage overhead can be avoided without increasing the running time of the algorithm.

Note: A BFS traversal can be analogously used to solve this problem.

4. Let $G = (V, E)$ be a directed acyclic graph with $V = \{0, 1, 2, \ldots, n-1\}$. An edge (i, j) is in E if and only if $0 \leqslant i < j \leqslant n-1$. Suppose that each edge $(i, j) \in E$ carries a cost $c(i, j)$. You should not assume that all $c(i, j)$ are positive (or non-negative). In other words, negative-cost edges are allowed. We want to solve the single-source-shortest-path (SSSP) problem with source $s = 0$.

(a) Deduce that G contains $\Theta(n^2)$ edges. **(3)**

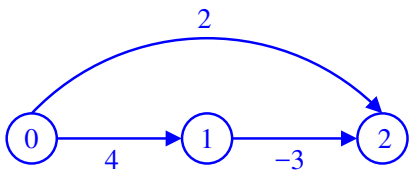
Solution For $i = 0, 1, 2, \ldots, n-1$, there are exactly $n-i-1$ edges from Vertex i, so $|E| = (n-1) + (n-2) + \cdots + 1 + 0 = n(n-1)/2 = \Theta(n^2)$.

(b) Prove that there are exactly 2^{i-1} directed paths from the source $s = 0$ to the vertex $i \in \{1, 2, 3, \ldots, n-1\}$. **(4)**

Solution Let $P(i)$ denote the number of $0, i$-paths in G. We have $P(0) = 1$, and $P(i) = \sum_{j=0}^{i-1} P(j)$ for $i \geqslant 1$ (take any $(0, j)$-path followed by the edge (j, i)). We prove by induction that $P(i) = \begin{cases} 1 & \text{if } i = 0, \\ 2^{i-1} & \text{if } i \geqslant 1. \end{cases}$ For $i \geqslant 1$, the inductive step can be proved as $P(i) = P(0) + \Big(P(1) + P(2) + \cdots + P(i-1)\Big) = 1 + \Big(1 + 2 + 2^2 + \cdots + 2^{i-2}\Big) = 2^{i-1}$.

(c) Prove/Disprove: Dijkstra's SSSP algorithm works for this G (in the presence of negative-cost edges). **(3)**

Solution The same counterexample given in the class ($n = 3$) shows that Dijkstra's SSSP algorithm may fail in this case.



(d) Irrespective of whether Dijkstra's SSSP algorithm works for G or not, it takes $O(n^2 \log n)$ running time in this case. Propose an $O(n^2)$-time algorithm to solve the SSSP problem in the given G (with source $s = 0$). **(10)**

Solution Let $D[j]$ store the shortest $0, j$ distance. Clearly, $D[0] = 0$. For setting $D[j]$ with $j \geqslant 1$, we follow a shortest $0, i$-path and then follow the edge (i, j). These costs are minimized over $i = 0, 1, 2, \ldots, j-1$. This gives the following dynamic-programming algorithm.

1. Set $D[0] = 0$ and $prev[0] = -$.
2. For $j = 1, 2, 3, \ldots, n-1$, repeat:
   (a) Let $i^* = \text{argmin}_{0 \leqslant i \leqslant j-1} \Big(D[i] + c(i, j) \Big)$.
   (b) Set $D[j] = D[i^*] + c(i^*, j)$.
   (c) Set $prev[j] = i^*$.

The *prev* array is used to construct shortest $0, t$-paths for all $t \in \{0, 1, 2, \ldots, n-1\}$. The path printed backward is $t, prev[t], prev[prev[t]], \ldots, 0$.

Step 1 takes $O(1)$ time. For a given j, Step 2 takes $O(j)$ time. So the total running time is $O(1 + 2 + \cdots + (n-1)) = O(n^2)$. Moreover, each shortest $0, t$-path can be printed in $O(n)$ time.

5. Let S be a string of lower-case letters *a–z*. A *blanagram* of S is obtained by changing a single letter of S, and then (optionally) permuting the symbols of this changed S. For example, let $S =$ *entrain*. Its blanagram *terrain* is obtained by changing one *n* to *r*, and then permuting the letters. Likewise, *trainer* is another blanagram of *entrain* (but not of *terrain*, because *trainer* and *terrain* consist of exactly the same letters).

Suppose that two strings S and T of the same length n and with the symbols in the range *a–z* are given as input (these need not be dictionary words). Your task is to determine whether S and T are blanagrams (of one another). Propose an $O(n)$-time algorithm to solve this problem. **(10)**

Solution The symbols of S and T come from an alphabet of constant size $k = 26$. We can run the first stage of counting sort to count the numbers of occurrences of different symbols in the two input strings. This takes $O(n+k) = O(n)$ running time. Checking for blanagrams can then be done in $O(k) = O(1)$ time. Here follows a C code snippet implementing this algorithm.

```
int blanagram ( int S[], int T[], int n )
{
    int i, k;
    int C[26], D[26];
    int plus1, minus1;

    for (k=0; k<26; ++k) C[k] = D[k] = 0;

    for (i=0; i<n; ++i) {
        k = S[i] - 'a'; ++(C[k]);
        k = T[i] - 'a'; ++(D[k]);
    }

    plus1 = minus1 = 0;
    for (k=0; k<26; ++k) {
        if (C[k] == D[k]) continue;
        if (C[k] == D[k] + 1) { ++plus1; continue; }
        if (C[k] == D[k] - 1) { ++minus1; continue; }
        return FALSE;
    }

    if ((plus1 == 1) && (minus1 == 1)) return TRUE;
    return FALSE;
}
```