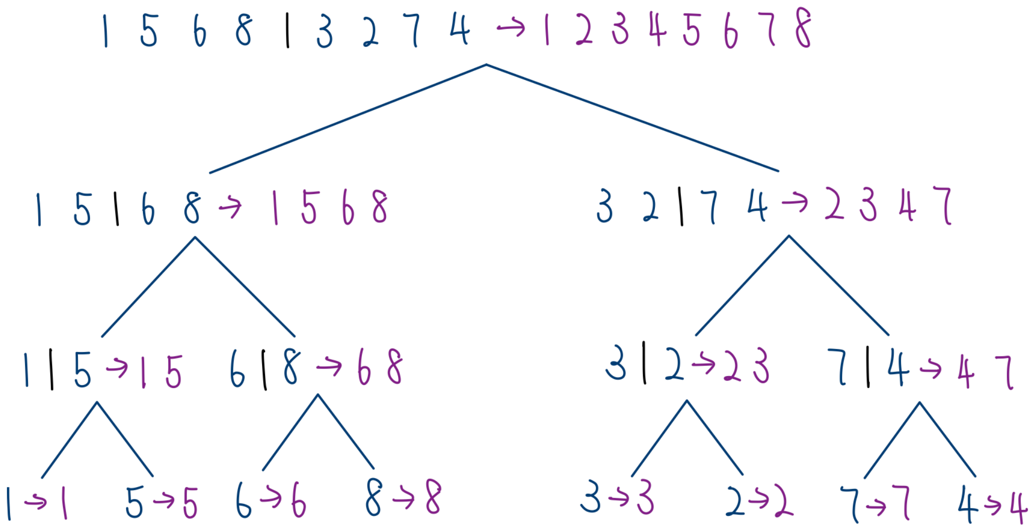
Q1. [12 marks] Let $A[0 \cdots 7] = [1, 5, 6, 8, 3, 2, 7, 4]$.

- **(i). [6 marks]** Show the process of **mergesort**(A, 0, 7) to sort A in ascending order step by step.
- **(ii). [6 marks]** Assume that we call **quicksort**(A, 0, 7) to sort A in ascending order, and the pivot position we randomly choose is 1. Show how the partition works and indicate the value of **nextsmallpos** during the invocation of **partition**(A, 0, 7, 1) step by step.

(i).



(ii). *Note: Symbol "↔" is used to denote the swap operation between 2 elements*

Since the pivot position = 1, the pivotVal = A[1] = 5.

Steps:

1. Swap the pivot $A[1]$ and $A[7]$. $A[0 \cdots 7] = [1, 4, 6, 8, 3, 2, 7, 5] \Rightarrow$ pivot = 7
2. When j = 0, nextsmallpos = 0
3. When j = 1, nextsmallpos = 1
4. When j = 2, nextsmallpos = 2
5. When j = 3, nextsmallpos = 2
6. When j = 4, nextsmallpos = 2. $A[4] \leftrightarrow A[2]$, $A[0 \cdots 7] = [1, 4, 3, 8, 6, 2, 7, 5]$
7. When j = 5, nextsmallpos = 3. $A[5] \leftrightarrow A[3]$, $A[0 \cdots 7] = [1, 4, 3, 2, 6, 8, 7, 5]$
8. When j = 6, nextsmallpos = 4.
9. Swap the pivot to nextsmallpos. $A[4] \leftrightarrow A[7]$, $A[0 \cdots 7] = [1, 4, 3, 2, 5, 8, 7, 6]$

$\therefore$ quicksort(A, 0, 7) will return [1, 4, 3, 2, 5, 8, 7, 6]

Q2. [12 marks] Sort array $A[1 \cdots 7] = [9, 1, 10, 3, 2, 8, 4]$ in decreasing order by heap sort. (you may just show the array representation at each step.)

- **(i). [6 marks]** Show the contents of A in the heap adjust process to make it a min-heap step by step.
- **(ii). [6 marks]** Using the min-heap of Part (i), show the contents of A in the sorting process of swaping elements in the array step by step.

(i). *Steps:*

1. Node id 1 violates the min-heap property. Swap it with its smaller child.
   So, $A[1 \cdots 7] = [1, 9, 10, 3, 2, 8, 4]$
2. Node id 2 violates the min-heap property. Swap it with its smaller child.
   So, $A[1 \cdots 7] = [1, 2, 10, 3, 9, 8, 4]$
3. Node id 3 violates the min-heap property. Swap it with its smaller child.
   So, $A[1 \cdots 7] = [1, 2, 4, 3, 9, 8, 10]$

(ii). *Note:*
Symbol "↔" denotes the swap operation between 2 elements
Bold elements are sorted

Steps:

1. Node id 7 ↔ Node id 1 and reduce the heap size by 1.
   $A[1 \cdots 7]$ = [10, 2, 4, 3, 9, 8, **1**]
2. Swap until A is a min heap: $A[1 \cdots 7]$ = [2, 3, 4, 10, 9, 8, **1**]
3. Node id 6 ↔ Node id 1 and reduce the heap size by 1.
   $A[1 \cdots 7]$ = [8, 3, 4, 10, 9, **2**, **1**]
4. Swap until A is a min heap: $A[1 \cdots 7]$ = [3, 8, 4, 10, 9, **2**, **1**]
5. Node id 5 ↔ Node id 1 and reduce the heap size by 1.
   $A[1 \cdots 7]$ = [9, 8, 4, 10, **3**, **2**, **1**]
6. Swap until A is a min heap: $A[1 \cdots 7]$ = [4, 8, 9, 10, **3**, **2**, **1**]
7. Node id 4 ↔ Node id 1 and reduce the heap size by 1.
   $A[1 \cdots 7]$ = [10, 8, 9, **4**, **3**, **2**, **1**]
8. Swap until A is a min heap: $A[1 \cdots 7]$ = [8, 10, 9, **4**, **3**, **2**, **1**]
9. Node id 3 ↔ Node id 1 and reduce the heap size by 1.
   $A[1 \cdots 7]$ = [9, 10, **8**, **4**, **3**, **2**, **1**]
10. Min-heap property satisfied. No change is needed.
11. Node id 2 ↔ Node id 1 and reduce the heap size by 1.
    $A[1 \cdots 7]$ = [10, **9**, **8**, **4**, **3**, **2**, **1**]
12. Now, the heap size = 1. The array is sorted, which $A[1 \cdots 7]$ = [**10**, **9**, **8**, **4**, **3**, **2**, **1**].

Q3. [14 marks] Assume that we have a hash table with size $m = 13$ and the hash function $h(k) = 2k\%13$. We use linear probing to address collisions. Answer the following questions.

- **(i). [7 marks]** Given an empty hash table, show the hash table when inserting 10, 2, 23, 13, 1, 9, 17 in order step by step.
- **(ii). [7 marks]** Given the following hash table, show the records examined when searching for 42.

0				4				8				12
0					22	16	10	30	24	42	25	

(i).

Insert 10, where $h(10) = 7$:

							10					

Insert 2, where $h(2) = 4$:

				2			10					

Insert 23, where $h(23) = 7$:

				2			10	23				

Insert 13, where $h(13) = 0$:

13				2			10	23				

Insert 1, where $h(1) = 2$:

13		1		2			10	23				

Insert 9, where $h(9) = 5$:

13		1		2	9		10	23				

Insert 17, where $h(17) = 8$:

13		1		2	9		10	23	17			

(ii). $\because$ $h(42) = 6$

$\therefore$ We will examine 16, 10, 30, 24 and last 42 is found.

Q4. [14 marks] Assume that we have a hash table with size $m = 13$ and we use double hashing to address collisions. The double hashing function is $h(k, i) = (h(k)+i \cdot h'(k))\%m$, where $h(k) = k\%13$ and $h'(k) = 1 + k\%3$. Answer the following questions.

- **(i). [7 marks]** Given an empty hash table, show the hash table when inserting 14, 2, 18, 36, 31, 23, 42 in order step by step.
- **(ii). [7 marks]** Given the following hash table, show the records examined when searching for 44.

0				4				8				12
		15		30	5			21		10	24	38

(i).

Insert 14, where $h(14,0) = 1$:

	14											

Insert 2, where $h(2,0) = 2$:

	14	2										

Insert 18, where $h(18,0) = 5$:

	14	2			18							

Insert 36, where $h(36,0) = 10$:

	14	2			18					36		

Insert 31, where $h(31,0) = 5$ and $h(31,1) = 7$:

	14	2			18		31			36		

Insert 23, where $h(23,0) = 10$ and $h(23,1) = 0$:

23	14	2			18		31			36		

Insert 42, where $h(42,0) = 3$:

23	14	2	42		18		31			36		

(ii). $\because$ $h(44,0) = 5$, $h(44,1) = 8$, $h(44,2) = 11$.

$\therefore$ We will examine 5, 21, 24, 38 and we stop searching as we meet the empty slot.

Q5. [38 marks] Consider the directed graph G_1 as shown in Fig. 1. Answer the following questions.

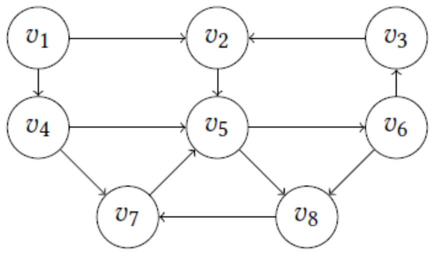


Fig. 1. Directed Graph G_1 for Q5

- **(i). [2 mark]** Calculate the out-degree and the in-degree of v_5.
- **(ii). [2 mark]** Whether the path $v_1 \to v_4 \to v_5 \to v_6 \to v_8 \to v_7$ is a simple path? Justify your answer.
- **(iii). [8 marks]** For G_1, show both its adjacency list representation and its adjacency matrix representation. (The nodes should be in ascending order of ID.)

(i). $d_{out}(v_5) = 2$, $d_{in}(v_5) = 3$

(ii). $\because$ All the nodes between the 1st node and the last node are distinct
$\therefore$ Yes

(iii). Adjacency list:

v_1: $h \leftrightarrow v_2 \leftrightarrow v_4 \leftrightarrow t$
v_2: $h \leftrightarrow v_5 \leftrightarrow t$
v_3: $h \leftrightarrow v_2 \leftrightarrow t$
v_4: $h \leftrightarrow v_5 \leftrightarrow v_7 \leftrightarrow t$
v_5: $h \leftrightarrow v_6 \leftrightarrow v_8 \leftrightarrow t$
v_6: $h \leftrightarrow v_3 \leftrightarrow v_8 \leftrightarrow t$
v_7: $h \leftrightarrow v_5 \leftrightarrow t$
v_8: $h \leftrightarrow v_7 \leftrightarrow t$

Adjacency matrix:

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_1	0	1	0	1	0	0	0	0
v_2	0	0	0	0	1	0	0	0
v_3	0	1	0	0	0	0	0	0
v_4	0	0	0	0	1	0	1	0
v_5	0	0	0	0	0	1	0	1
v_6	0	0	1	0	0	0	0	1
v_7	0	0	0	0	1	0	0	0
v_8	0	0	0	0	0	0	1	0

– **(iv). [8 marks]** Traverse G_1 using breadth-first search with v_1 as the source, assuming that the out-neighbors of a node are visited in ascending order of ID. Show the process and the contents of queue Q step by step. You may use 0 to denote the color to be white, 1 to denote the color to be gray, and 2 to denote the color to be black.
– **(v). [8 marks]** According to the results of Part (iv), show the contents of **minlength** array and **prev** array respectively.
– **(vi). [4 marks]** Show how to get the minimum length path from the source v_1 to v_7 using the **minlength** array and **prev** array. Justify your answer.
– **(vii). [6 marks]** Draw the BFS tree of Part (iv).

(iv). *Steps:*

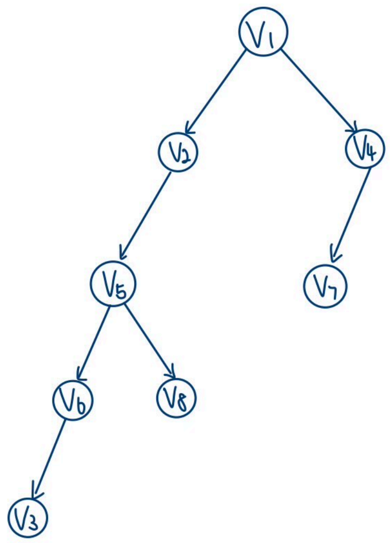
1. v_1:0, v_2:0, v_3:0, v_4:0, v_5:0, v_6:0, v_7:0, v_8:0
2. v_1:1, v_2:0, v_3:0, v_4:0, v_5:0, v_6:0, v_7:0, v_8:0, $Q = (v_1)$
3. v_1:2, v_2:1, v_3:0, v_4:1 v_5:0, v_6:0, v_7:0, v_8:0, $Q = (v_2, v_4)$
4. v_1:2, v_2:2, v_3:0, v_4:1 v_5:1, v_6:0, v_7:0, v_8:0, $Q = (v_4, v_5)$
5. v_1:2, v_2:2, v_3:0, v_4:2 v_5:1, v_6:0, v_7:1, v_8:0, $Q = (v_5, v_7)$
6. v_1:2, v_2:2, v_3:0, v_4:2 v_5:2, v_6:1, v_7:1, v_8:1, $Q = (v_7, v_6, v_8)$
7. v_1:2, v_2:2, v_3:0, v_4:2 v_5:2, v_6:1, v_7:2, v_8:1, $Q = (v_6, v_8)$
8. v_1:2, v_2:2, v_3:1, v_4:2 v_5:2, v_6:2, v_7:2, v_8:1, $Q = (v_8, v_3)$
9. v_1:2, v_2:2, v_3:1, v_4:2 v_5:2, v_6:2, v_7:2, v_8:2, $Q = (v_3)$
10. v_1:2, v_2:2, v_3:2, v_4:2 v_5:2, v_6:2, v_7:2, v_8:2, $Q = ()$

(v).

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
minlength	0	1	4	1	2	3	2	3
prev	*nil*	v_1	v_6	v_1	v_2	v_5	v_4	v_5

(vi). Previous node of v_7 is v_4, and the previous node of v_4 is v_1, which is the source node.
∴ The path from v_1 to v_7: $v_1 \rightarrow v_4 \rightarrow v_7$
∴ The minimum path length = 3

(vii).



Q6. [10 marks] Given an undirected graph $G = (V, E)$, A triangle consists of three nodes in G that are pairwise adjacent. More formally, a triangle in G is triplet (u, v, w) where $u, v, w \in V$ and $(u, v), (v, w), (w, u) \in E$. Consider the undirected graph G_2 as shown in Fig. 2. There are two triangles in G_2, (v_0, v_1, v_2) and (v_1, v_2, v_3).

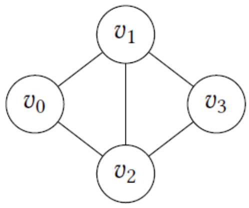


Fig. 2. An Undirected Graph G_2 for Q6

- **(i). [4 marks]** Please design an algorithm TriangleCounting(G) using pseudo-code with the provided Graph ADT, to count the number of triangles in G.

> Graph **ADT**.
> - **`Vertices(G)`:** Lists all vertices u in G.
> - **`Neighbors(G,u)`:** Lists all vertices v such that there is an edge between the vertex u and the vertex v.
> - **`Adjacent(G,u,v)`:** Tests whether there is an edge between the vertex u and the vertex v.

(ii). [2 marks] Assume that G has n nodes and m edges, and the degree of any node u in G is smaller than d_{max}. What is the time complexity of TriangleCounting(G) expressed in terms of n, m, and d_{max} if the graph ADT is implemented using an adjacency matrix? Justify your answer.

(i).

```
TriangleCounting(G):
    count = 0
    vertices <- Vertices(G)
    for each u in vertices:
        neighbors = Neighbors(G, u)
        for each v, w in neighbors:
            if Adjacent(G, v, w):
                count++
    return count
```

(ii). For the adjacency matrix:

`Neighbors(G, u)` costs $O(n)$ and `Adjacent(G, v, w)` costs $O(1)$.

$\therefore$ Outer for-loop costs $O(n^2)$ and inner for-loop cost $O(n)$

$\therefore$ Time complexity for `TriangleCounting(G)`= $O(n^3)$

(iii). [2 marks] Assume that G has n nodes and m edges, and the degree of any node u in G is smaller than d_{max}. What is the time complexity of TriangleCounting(G) expressed in terms of n, m, and d_{max} if the graph ADT is implemented using an adjacency list? Justify your answer.

(iv). [2 marks] Assume that the graph ADT is implemented using an adjacency list, and the output of Neighbors(G, u) is guaranteed to be sorted in ascending order by node ID. Please design a more efficient TriangleCounting(G) using pseudo-code.

(iii). For the adjacency list:

`Neighbors(G, u)` costs $O(d_{max})$ and `Adjacent(G, v, w)` costs $O(d_{max})$.

$\therefore$ Outer for-loop costs $O(nd_{max})$ and inner for-loop cost $O(nd_{max})$

$\therefore$ Time complexity for `TriangleCounting(G)`$= O((nd_{max})^2)$

(iv).

```
TriangleCounting(G):
  count = 0
  vertices <- Vertices(G)
  for u in vertices:
     neighbors_u = Neighbors(G, u)
     n = 0, m = 0;
     while n < length(neighbors_u) and m < length(neighbors_u):
        v = neighbors_u[n]
        w = neighbors_u[m]
        if v < w:
           n++
        else if v > w:
           m++
        else:
           m++
           while m < length(neighbors_u) and neighbors_u[m] == w:
              m++
              count += (n - m - 1) / 2
  return count
```