Revision

Final Test

- Test 2 is scheduled for **Week 14, Tue, on Dec 3, 2024**, with two sessions:
  - 9:00am-10:30am (T01 to T17, T38, T39)
  - 10:40am-12:10pm (T18 to T37)
- Venue:**E3-01-01 Lectorial 8**
- It will be a closed-book, written test: MCQ, T/F, Short Answer
- You are permitted a double-sided, handwritten A4 note. You must turn in this note with your exam.
- Calculators are also permitted but not necessary.
- Write within the answer box.
- Bring your SIT card.
- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps to earn partial credit.
- Consultation Session: Thu, 3:00pm-5:00pm E1-L1-MR203

MCQ

Non-Inversion is a pair (i, j), where 0 <= i < j < nums.length and nums[i] <= nums[j]. How many Non-Inversion are there in array [2,4,3,5,1]?

A. 4

B. 5

C. 7

D. 6

MCQ

Non-Inversion is a pair (i, j), where 0 <= i < j < nums.length and nums[i] <= nums[j]. How many Non-Inversion are there in array [2,4,3,5,1]?

A. 4

B. 5

C. 7

D. 6

MCQ

Which of the following is NOT a valid way to improve the time complexity of a divide-and-conquer solution to a problem?

A. Preprocessing

B. Reducing the number of subproblems

C. Debug to resolve bugs

D. Pruning

MCQ

Which of the following is NOT a valid way to improve the time complexity of a divide-and-conquer solution to a problem?

A. Preprocessing

B. Reducing the number of subproblems

C. Debug to resolve bugs

D. Pruning

MCQ

In the code of 3-way Partitioning Algorithm (National Flag Problem)

What are the missing lines for Line 1-2?

A. Line1:high--;mid--; Line2:mid++;

B. Line1:high--;mid--; Line2:mid--;

C. Line1:high--; Line2:mid++;

D. Line1:high--; Line2:mid--;

```
void threeWayPartition(vector<int>& arr, int pivot) {
    int low = 0;              // Pointer for elements < pivot
    int mid = 0;              // Pointer for current element
    int high = arr.size() - 1; // Pointer for elements > pivot
    while (mid <= high) {
        if (arr[mid] < pivot) {
            swap(arr[mid], arr[low]);
            low++;
            mid++;
        } else if (arr[mid] > pivot) {
            swap(arr[mid], arr[high]);
            Line 1:
        } else {
            Line 2:
        }
    }
}
```

MCQ

In the code of 3-way Partitioning Algorithm (National Flag Problem)

What are the missing lines for Line 1-2?

A. Line1:high--;mid--; Line2:mid++;

B. Line1:high--;mid--; Line2:mid--;

C. Line1:high--; Line2:mid++;

D. Line1:high--; Line2:mid--;

```
void threeWayPartition(vector<int>& arr, int pivot) {
    int low = 0;              // Pointer for elements < pivot
    int mid = 0;              // Pointer for current element
    int high = arr.size() - 1; // Pointer for elements > pivot
    while (mid <= high) {
        if (arr[mid] < pivot) {
            swap(arr[mid], arr[low]);
            low++;
            mid++;
        } else if (arr[mid] > pivot) {
            swap(arr[mid], arr[high]);
            Line 1: high--;
        } else {
            Line 2: mid++;
        }
    }
}
```

MCQ

- Perform DFS on a Graph <V, E>, n is the number of nodes and m is the number of edges. The time complexity is:
  A. O(m)
  B. O(n)
  C. O(m+n)
  D. O(mlog(n))

MCQ

- Perform DFS on a Graph <V, E>, n is the number of nodes and m is the number of edges. The time complexity is:
  A. O(m)
  B. O(n)
  C. O(m+n)
  D. O(mlog(n))

MCQ

- Which algorithm has a time complexity of $O(n^2)$ on a fully connected graph? n is the number of nodes.

A. Bellman-ford algorithm

B. Prim's algorithm with binary heap implementation.

C. Dijkstra algorithm with binary heap implementation.

D. BFS algorithm.

MCQ

- Which algorithm has a time complexity of $O(n^2)$ on a fully connected graph? n is the number of nodes.

A. Bellman-ford algorithm

B. Prim's algorithm with binary heap implementation.

C. Dijkstra algorithm with binary heap implementation.

D. BFS algorithm.

True or False

- Given an array A containing n comparable items, sort A using merge sort. While sorting, each item in A is compared with O(log n) other items of A.

- For a connected, weighted graph with n vertices and exactly n edges, it is possible to find a minimum spanning tree in O(n) time.

- A* and Dijkstra's Algorithm could produce the same shortest path in a graph.

True or False

- Given an array A containing n comparable items, sort A using merge sort. While sorting, each item in A is compared with O(log n) other items of A.
  - False. During the final merge step between two sorted halves of the array, each of size O(n) items, one item may be compared with O(n) items.

- For a connected, weighted graph with n vertices and exactly n edges, it is possible to find a minimum spanning tree in O(n) time.
  - True. This graph only contains one cycle, which can be found by a DFS. Just remove the heaviest edge in that cycle.

- A* and Dijkstra's Algorithm could produce the same shortest path in a graph.
  - True. If the A* heuristic function is carefully designed, admissible (never overestimates the true cost) and consistent (obeys the triangle inequality).

Short Answer

Design an algorithm to find the smallest k numbers in an array. The k numbers can be returned in any order as long as they satisfy the condition. You can write in C/C++ pseudo code for the implementation. What is the (expected) time complexity?

Example:

Input:

arr = [1, 3, 5, 7, 2, 4, 6, 8], k = 4

Output:

[1, 2, 3, 4]


```
vector<int> findSmallestKNumbers(vector<int>& arr, int k) {
    if (k <= 0 || arr.empty()) return {};
    quickSelect(arr, 0, arr.size() - 1, k);
    return vector<int>(arr.begin(), arr.begin() + k);
}
void quickSelect(vector<int>& arr, int left, int right, int k) {
    if (left >= right) return;
    // Use the randomized partition function
    int pivotIndex = randomizedPartition(arr, left, right);
    // Number of elements on the left including the pivot
    int count = pivotIndex - left + 1;
    if (count == k)
        return;
    else if (count < k) {
        k -= count;  // Nemaining elements needed
        quickSelect(arr, pivotIndex + 1, right, k);
    } else {
        quickSelect(arr, left, pivotIndex - 1, k);
    }
}
```

```
int randomizedPartition(vector<int>& arr, int left, int right) {
    int randomIndex = left + rand() % (right - left + 1);
    swap(arr[randomIndex], arr[right]);
    return partition(arr, left, right);
}
int partition(vector<int>& arr, int left, int right) {
    int pivot = arr[right];
    int i = left;
    for (int j = left; j < right; ++j) {
        if (arr[j] < pivot) {
            swap(arr[i], arr[j]);
            i++;
        }
    }
    swap(arr[i], arr[right]);
    return i;
}
```

Short Answer

You are given an integer array cost where cost[i] is the cost of ith step on a staircase. Once you pay the cost, you can either climb one or two steps. You can either start from the step with index 0, or the step with index 1. Return the minimum cost to reach the top of the floor. You can write in C/C++ pseudo code for the implementation. What is meaning of you dp function or table, what is the (expected) time complexity?

Example 1:
Input: cost = [10,15,20]
Output: 15
Explanation: You will start at index 1.
- Pay 15 and climb two steps to reach the top.
The total cost is 15.

Example 2:
Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: You will start at index 0.
- Pay 1 and climb two steps to reach index 2.
- Pay 1 and climb two steps to reach index 4.
- Pay 1 and climb two steps to reach index 6.
- Pay 1 and climb one step to reach index 7.
- Pay 1 and climb two steps to reach index 9.
- Pay 1 and climb one step to reach the top.
The total cost is 6.

- dp[i]: the minimum cost to climb to i from 0 or 1

```
int minCostClimbingStairs(vector<int> &cost) {
        int n = cost.size();
        vector<int> dp(n + 1);
        for (int i = 2; i <= n; i++) {
            dp[i] = min(dp[i - 1] + cost[i - 1], dp[i - 2] + cost[i - 2]);
        }
        return f[n];
    }
```

Time analysis: O(n), n is the size of cost

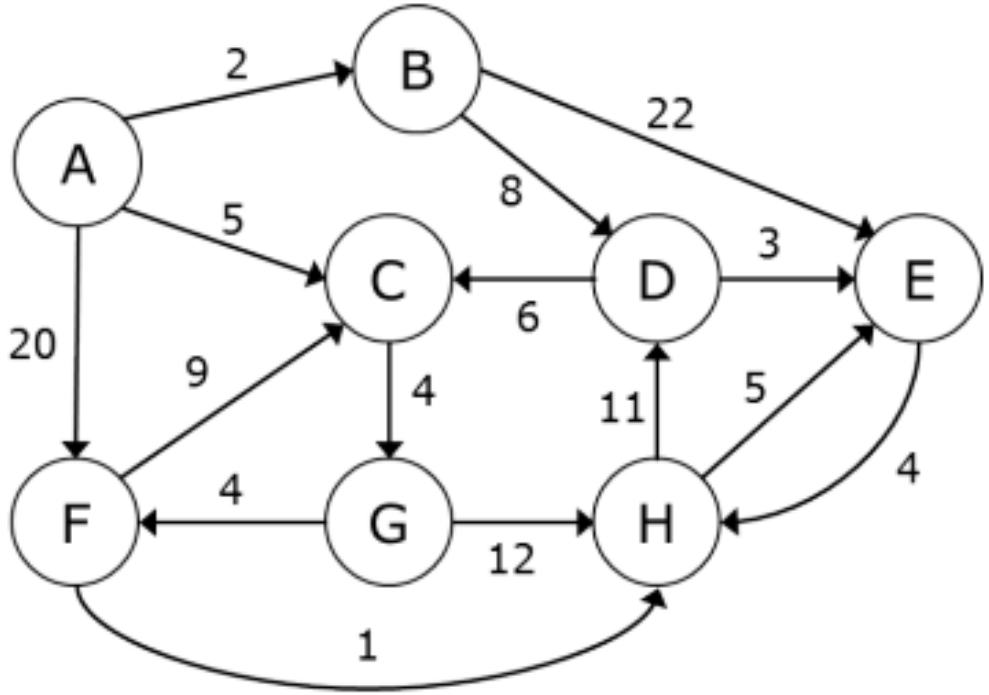
Short Answer

Consider the following directed, weighted graph:

Use Dijkstra's algorithm to calculate the single-source shortest paths from vertex A to every other vertex. Show your steps in the table below. As the algorithm proceeds, cross out old values and write in new ones, from left to right in each cell. If during your algorithm two unvisited vertices have the same distance, use alphabetical order to determine which one is selected first. Also list the vertices in the order which Dijkstras algorithm marks them visited:

(a) Order vertices marked as visited: _____

Vertex	Visited	dist	edgeTo



(b) What is the lowest-cost path from A to H in the graph, as computed above?

Short Answer

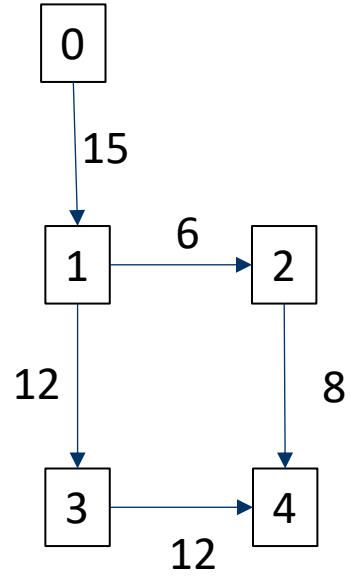
(a) Order vertices marked as visited: A B C G D E F H

Vertex	Visited	dist	edgeTo
A	Y	0	-
B	Y	2	A
C	Y	5	A
D	Y	10	B
E	Y	~~24~~ 13	D
F	Y	~~20~~ 13	G
G	Y	9	C
H	Y	~~17~~ 14	F

(b) A–C–G–F–H

Short Answer

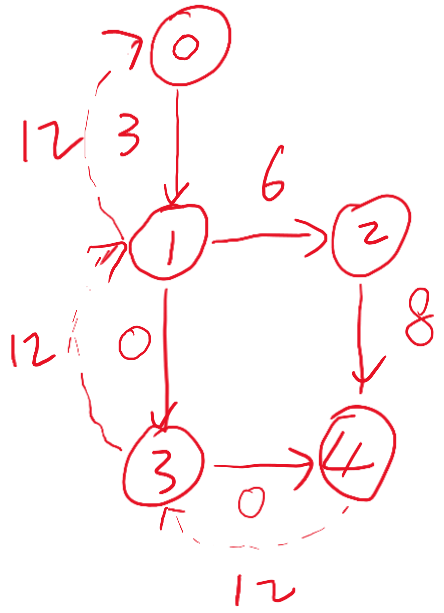
- Given the following directed graph, where each edge is labeled with its capacity:
- Compute the maximum flow from the source 0 to the sink 4 using the **Ford-Fulkerson algorithm**. Show the steps clearly, including:
  - The residual graph.
  - The augmenting paths found at each iteration.
  - The flow adjustments.
  - The final maximum flow value.



Short Answer

- Max flow is 15

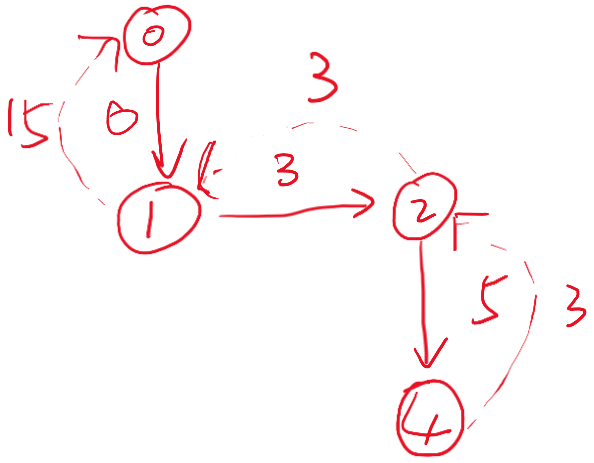
Bottleneck : 12



Path : 0 -> 1 -> 3 -> 4

Flow : 12

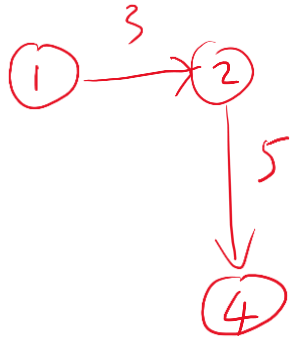
Bottleneck : 3



Path : 0 -> 1 -> 2 -> 4

Flow : 3

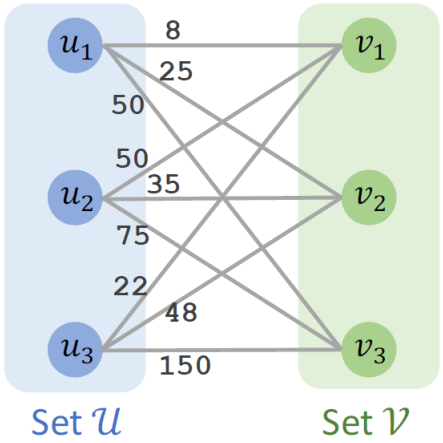
Residual



max flow is 15

Short Answer

- Apply Hungarian Algorithm to solve the assignment problem for this bipartite graph:



- Show each step of the algorithm (result after row reduction, result after column reduction, iterations of covering zeros (showing the lines, and the result of each iteration), output the matching).

- Subtract Row Minima

	v_1	v_2	v_3
u_1	8	25	50
u_2	50	35	75
u_3	22	48	150

	v_1	v_2	v_3
u_1	0	17	42
u_2	15	0	40
u_3	0	26	128

- Subtract Column Minima

	v_1	v_2	v_3
u_1	0	17	2
u_2	15	0	0
u_3	0	26	88

- Iteration 1

	v_1	v_2	v_3
u_1	0	17	2
u_2	~~15~~	~~0~~	~~0~~
u_3	0	26	88

	v_1	v_2	v_3
u_1	0	15	0
u_2	17	0	0
u_3	0	24	86

- Iteration 2

	v_1	v_2	v_3
u_1	~~0~~	~~15~~	~~0~~
u_2	~~17~~	~~0~~	~~0~~
u_3	0	24	86

- Output the matching