

### AI 3: GREEDY SEARCH ALGORITHMS

#### Assign 3A : SELECTION SORT ALGORITHM

```
def selectionSort( itemList ):
    n = len( itemList )
    for i in range( n - 1 ):
        minValueIndex = i

        for j in range( i + 1, n ):
            if itemList[j] < itemList[minValueIndex] :
                minValueIndex = j

        if minValueIndex != i :
            temp = itemList[i]
            itemList[i] = itemList[minValueIndex]
            itemList[minValueIndex] = temp

    return itemList


el = [21,6,9,33,3]

print(selectionSort(el))
```

Programiz




Online Compiler

Courses by Programiz



Programiz PRO

main.py

 Share

Run

Output

```
# Online Python compiler (interpreter) to run Python online.
def selectionSort( itemList ):
    n = len( itemList )
    for i in range( n - 1 ):
        minValueIndex = i

        for j in range( i + 1, n ):
            if itemList[j] < itemList[minValueIndex] :
                minValueIndex = j

        if minValueIndex != i :
            temp = itemList[i]
            itemList[i] = itemList[minValueIndex]
            itemList[minValueIndex] = temp

    return itemList

e1 = [21,6,9,33,3]

print(selectionSort(e1))
```

```
[3, 6, 9, 21, 33]

=== Code Execution Successful ===
```

## AI 3: GREEDY SEARCH ALGORITHMS

### Assign 3B

#### 2. Single Source Shortest Path (Dijkstra's Algorithm)

```
# Dijkstra's Algorithm
import heapq

def dijkstra(graph, start):
    n = len(graph)
    distances = [float('inf')] * n
    distances[start] = 0
    pq = [(0, start)]

    while pq:
        current_dist, u = heapq.heappop(pq)
        if current_dist > distances[u]:
            continue
        for v, weight in graph[u]:
            distance = current_dist + weight
            if distance < distances[v]:
                distances[v] = distance
                heapq.heappush(pq, (distance, v))
    return distances

# Example usage:
graph_dijkstra = {
    0: [(1, 4), (2, 1)],
    1: [(3, 1)],
```

```

    2: [(1, 2), (3, 5)],
    3: []
}

start_node = 0

distances = dijkstra(graph_dijkstra, start_node)

print(distances)

```

The screenshot shows an online compiler interface with a dark theme. At the top, there's a banner for 'Online Compiler' and a prompt to 'Choose the right AI'. Below the banner, the code editor displays a Python script for Dijkstra's algorithm. The script defines a priority queue, a while loop to process nodes, and a graph dictionary. The output panel on the right shows the result of the execution: '[0, 3, 1, 4]' and a success message.

```

n.py
pq = [(0, start)]

while pq:
    current_dist, u = heapq.heappop(pq)
    if current_dist > distances[u]:
        continue
    for v, weight in graph[u]:
        distance = current_dist + weight
        if distance < distances[v]:
            distances[v] = distance
            heapq.heappush(pq, (distance, v))
    return distances

# Example usage:
graph_dijkstra = {
    0: [(1, 4), (2, 1)],
    1: [(3, 1)],
    2: [(1, 2), (3, 5)],
    3: []
}

```

Output: [0, 3, 1, 4]

=== Code Execution Successful ===

## AI 3: GREEDY SEARCH ALGORITHMS

### Assign 3C

#### 3. Kruskal's Minimum Spanning Tree

# Kruskal's Algorithm

class DisjointSet:

def \_\_init\_\_(self, n):

self.parent = list(range(n))

def find(self, x):

if self.parent[x] != x:

self.parent[x] = self.find(self.parent[x])

return self.parent[x]

def union(self, x, y):

x\_root = self.find(x)

y\_root = self.find(y)

if x\_root != y\_root:

self.parent[x\_root] = y\_root

return True

return False

def kruskal(n, edges):

edges.sort(key=lambda x: x[2])

ds = DisjointSet(n)

mst = []

total\_weight = 0

for u, v, weight in edges:

```
if ds.union(u, v):  
    mst.append((u, v, weight))  
    total_weight += weight
```

```
return mst, total_weight
```

# Example usage:

```
edges_kruskal = [  
    (0, 1, 10),  
    (0, 2, 6),  
    (0, 3, 5),  
    (1, 3, 15),  
    (2, 3, 4)  
]  
  
mst_k, weight_k = kruskal(4, edges_kruskal)  
print(mst_k, weight_k)
```



The screenshot shows a web-based Python compiler interface. The top bar includes a 'Programiz' logo, a 'Python Online Compiler' title, and several utility buttons like 'Set Password', 'Share it securely.', 'Learn more', 'Acrobat', and 'Programiz PRO'. Below the bar, the editor area is titled 'main.py' and contains the following Python code:

```
1 class DisjointSet:  
2     def __init__(self, n):  
3         self.parent = list(range(n))  
4  
5     def find(self, x):  
6         if self.parent[x] != x:  
7             self.parent[x] = self.find(self.parent[x])  
8         return self.parent[x]  
9  
10    def union(self, x, y):  
11        x_root = self.find(x)  
12        y_root = self.find(y)  
13        if x_root != y_root:  
14            self.parent[x_root] = y_root  
15        return True  
16        return False  
17  
18 def kruskal(n, edges):  
19     edges.sort(key=lambda x: x[2])  
20     ds = DisjointSet(n)  
21     mst = []  
22     total_weight = 0  
23  
24     for u, v, weight in edges:
```

The right side of the interface shows the 'Output' panel, which displays the result of the code execution: `[(2, 3, 4), (0, 3, 5), (0, 1, 10)] 19`. Below the output, it states '=== Code Execution Successful ==='. The 'Run' button is highlighted in blue, and a 'Clear' button is visible in the top right of the output panel.

## AI 3: GREEDY SEARCH ALGORITHMS

### Assign 3D

#### 4. Prim's Minimum Spanning Tree

# Prim's Algorithm

```
def prim(graph):
```

```
    n = len(graph)
```

```
    visited = [False] * n
```

```
    min_heap = [(0, 0)]
```

```
    total_cost = 0
```

```
    mst = []
```

```
    while min_heap:
```

```
        weight, u = heapq.heappop(min_heap)
```

```
        if visited[u]:
```

```
            continue
```

```
        visited[u] = True
```

```
        total_cost += weight
```

```
        for v, w in graph[u]:
```

```
            if not visited[v]:
```

```
                heapq.heappush(min_heap, (w, v))
```

```
                mst.append((u, v, w))
```

```
    return mst, total_cost
```

# Example usage:

```
graph_prim = {
```

```
    0: [(1, 10), (2, 6), (3, 5)],
```

```
    1: [(0, 10), (3, 15)],
```

2:  $[(0, 6), (3, 4)]$ ,

3:  $[(0, 5), (1, 15), (2, 4)]$

}

```
mst_p, weight_p = prim(graph_prim)
```

```
print(mst_p, weight_p)
```

```
main.py  Run  Output
1 # Online Python compiler (interpreter) to run Python online
2 import heapq  # Required for using heapq
3
4 # Prim's Algorithm
5 def prim(graph):
6     n = len(graph)
7     visited = [False] * n
8     min_heap = [(0, 0, -1)] # (weight, current_vertex, from_vertex)
9     total_cost = 0
10    mst = []
11
12    while min_heap:
13        weight, u, parent = heapq.heappop(min_heap)
14        if visited[u]:
15            continue
16        visited[u] = True
17        total_cost += weight
18        if parent != -1:
19            mst.append((parent, u, weight))
20        for v, w in graph[u]:
21            if not visited[v]:
22                heapq.heappush(min_heap, (w, v, u))
23
24    return mst, total_cost
```

Minimum Spanning Tree edges: [(0, 3, 5), (3, 2, 4), (0, 1, 10)]  
Total Weight of MST: 19

=== Code Execution Successful ===



## AI 3: GREEDY SEARCH ALGORITHMS

### Assign 3E

#### 5. Job Scheduling Problem

##### # Job Scheduling Problem

```
class Job:
```

```
    def __init__(self, job_id, deadline, profit):
```

```
        self.id = job_id
```

```
        self.deadline = deadline
```

```
        self.profit = profit
```

```
def job_scheduling(jobs):
```

```
    jobs.sort(key=lambda x: x.profit, reverse=True)
```

```
    n = max(job.deadline for job in jobs)
```

```
    result = [None] * n
```

```
    slot = [False] * n
```

```
    total_profit = 0
```

```
    job_sequence = []
```

```
    for job in jobs:
```

```
        for j in range(min(n, job.deadline) - 1, -1, -1):
```

```
            if not slot[j]:
```

```
                slot[j] = True
```

```
                result[j] = job.id
```

```
                total_profit += job.profit
```

```
                job_sequence.append(job.id)
```

```
                break
```

```
return job_sequence, total_profit
```

```
# Example usage:
```

```
jobs_list = [
    Job('J1', 2, 100),
    Job('J2', 1, 19),
    Job('J3', 2, 27),
    Job('J4', 1, 25),
    Job('J5', 3, 15)
]
```

```
seq, prof = job_scheduling(jobs_list)
```

```
print(seq, prof)
```

main.py

Share

Run

Output

```
1 # Online Python compiler (interpreter) to run Python online.  
2 # Job Scheduling Problem  
3 class Job:  
4     def __init__(self, job_id, deadline, profit):  
5         self.id = job_id  
6         self.deadline = deadline  
7         self.profit = profit  
8  
9     def job_scheduling(jobs):  
10        jobs.sort(key=lambda x: x.profit, reverse=True)  
11        n = max(job.deadline for job in jobs)  
12        result = [None] * n  
13        slot = [False] * n  
14        total_profit = 0  
15        job_sequence = []  
16  
17        for job in jobs:  
18            for j in range(min(n, job.deadline) - 1, -1, -1):  
19                if not slot[j]:  
20                    slot[j] = True  
21                    result[j] = job.id  
22                    total_profit += job.profit  
23                    job_sequence.append(job.id)
```

['J1', 'J3', 'J5'] 142  
  
=== Code Execution Successful ===