

<Short Answer Questions, Part1>

For each of the following algorithms, give:

- a) the worst-case complexity
- b) the average case complexity of these algorithms

Give your answer in Big Oh notation.

If there is no difference in the worst case and the average case, then you may write "Same" for part b), the average case

1. Selection Sort of N items

- a) $O(N^2)$
- b) $O(N^2)$

2. Insertion Sort of N items

- a) $O(N^2)$
- b) $O(N^2)$

3. A sequence of M union-find operations on N disjoint sets.

Assume that the union operation is weighted quick union, and all operations use path compression.

- a) $O(M \log^* N)$
- b) $O(M \log^* N)$

4. The total copying cost when using the doubling algorithm for a sequence of N push operations. The initial size of stack is 1 element.

- a) $O(N)$
- b) $O(N)$

5. Quicksort, where the pivot is always 1st element, of N items

- a) $O(N^2)$
- b) $O(N \log N)$

6. Merge sort of N items

- a) $O(N \log N)$
- b) $O(N \log N)$

7. Quickselect, where pivot is always 1st element, to find the median of N items (assume N is an arbitrary odd number)

- a) $O(N^2)$
- b) $O(N)$

8. Quickselect, where pivot is always 1st element, to find the 4th smallest element of N items ($N \geq 4$).

- a) $O(N^2)$
- b) $O(N)$

9. Delete 1 element from a hash table that has N elements.

- a) $O(N)$
- b) $O(1)$

10. Find an item in a hash table that has N items.

- a) $O(N)$
- b) $O(1)$

<Short Answer Questions, cont>

Give the worst case performance (in Big Oh notation) for the following algorithmic problems

11. A LSB radix sort over an alphabet of size A, for fixed-length strings of length L. There are N such strings.

Ans: $O(NL)$

12. Build a binary heap from an array of size N.

Ans: $O(N)$

13. Find an item in a left-leaning red-black tree of size N.

Ans: $O(\log N)$

14. The 1st find operation in sequence of M Union/Find operations on N disjoint sets.

Ans: $O(1)$

15. For a graph $G = (V, E)$ with no negative edge weights, how long does Dijkstra's algorithm take to find the shortest path between distinguished vertices s and t?

Ans: $O(E \log V)$

16. Strassen's algorithm to multiply 2 square matrices of size $N \times N$?

Ans: $O(N^{\log_2 7})$

17. Given a sample string of length N, and a fixed pattern (NOT a regular expression) of length M, what is the complexity of searching the sample string for the pattern?

Ans: $O(M+N)$

18. Given a sample string of length N, and a (fully parenthesized) regular expression of length M, what is the complexity of searching the sample string for the pattern?

Ans: $O(MN)$

19. Given a graph $G = (V, E)$ with negative edge weights, but no negative cycles, what is the best known asymptotic complexity for finding the shortest path from 1 distinguished vertex s to all other vertices in the graph?

Ans: $O(VE)$

20. Given a graph $G = (V, E)$, with weighted edges given by the function $w(e)$ for $e \in E$, what is the complexity to find the minimum cost spanning tree for G?

Ans: $O(E \log V)$

<"Long" Answer Questions>

1. [4 pts] What is the complexity of the following program? (Big Oh notation is sufficient). Assume that $X[i][j]$ references the (i,j) component of matrix X . Also, assume that N parameter describing the size of X as $X[N,N]$, size of v as $v[N]$, and the size of b as $b[N]$.

```
for i in range(N+1):  
    v[i] = 0  
    for j in range(i+1,N+1,2):  
        v[i] += A[i][j]*b[j]
```

Ans: The complexity of this program is $O(N^2)$, the outer loop iterates $N+1$ times, and the inner loop iterates about $N/2$ times for each iteration of the outer loop, so the overall time complexity of the program is $O(N^2)$.

2. [6 pts] Suppose someone you know implemented Kruskal's algorithm for the minimum spanning tree of a graph $G = \{V, E\}$. Unfortunately, your acquaintance did not implement weighted union or path compression for the union/find operations. What is the complexity of this variant of Kruskal?

Ans: This variant of Kruskal's algorithm complexity will be $O(E \log V)$, where E is the number of edges and V is the number of vertices in the graph. Because the union/find operations will take $O(V)$ time to execute, and while loop will take $O(V^2)$, than building the Priority-Queue will take $O(E \log V)$. Thus, the Overall complexity is $O(E \log V)$.

3. [10 pts] Suppose you have an implementation of an A-heap. The A-heap is a heap-like data structure that has $O(1)$ complexity for the decrease-key operation. All other operations of the data structure have the same complexity as a traditional binary heap. What is the complexity of Dijkstra's algorithm using the A-heap data structure?

Ans: The complexity of Dijkstra's algorithm using the A-heap data structure is the same as using a traditional binary heap, which is $O((V + E) \log V)$, the decrease-key operation is used to update the priority of a vertex in the heap, and with the A-heap data structure, this operation has $O(1)$ complexity, the algorithm still has to perform other operations, such as inserting and extracting elements, which have the same complexity as in a binary heap.

4. [6 pts] Using any method you like, find the Big Oh complexity of the recurrence

$$T(n) = 5T\left(\frac{n}{4}\right) + \frac{n^3}{\log^2(n)}$$

Ans: We can use Master Theorem to find Big Oh complexity,

$$a = 5, b = 4, k = 3, p = -2$$

According to Master Theorem: $a < b^k$, $p < 0$

complexity will be $O(N^k) = O(N^3)$

Thus, The overall complexity of this recurrence is $O(N^3)$

5. [10 pts] You are given a complete, undirected graph $G = (V, E)$ with edge weights. A complete graph is a graph s.t. for any 2 distinct vertices v_1, v_2 , there is an edge $\langle v_1, v_2 \rangle \in E$. You construct a minimum cost spanning tree (MST) for G . Now suppose that 1 of the edges of the MST is deleted.

Give an efficient algorithm to construct a new MST from the old 1 without using the deleted edge. In addition to your algorithm, give the complexity of your algorithm.

My Python code & explain: The complexity of my algorithm is $O(N * \log N)$, where N is the number of nodes in the graph. Because each edge is visited once and the union-find operation has a logarithmic time complexity.

```
1 # Hsuan-You Lin Final Exam Question 5.
2 from collections import defaultdict
3 def find(parent, i):
4     if i < 0 or i >= len(parent):
5         return None
6     if parent[i] == i:
7         return i
8     return find(parent, parent[i])
9
10 # Function to construct a new MST from the old one without using the deleted edge
11 def new_mst(n, edges, deleted_edge):
12     # Create a disjoint set with n nodes
13     parent = [i for i in range(n + 1)]
14     rank = [0 for i in range(n + 1)]
15     min_cost = defaultdict(int)
16
17     for u, v, weight in edges:
18         if (u, v) == deleted_edge:
19             continue
20
21         x = find(parent, u)
22         y = find(parent, v)
23
24         # If the nodes are not in the same set, add the edge to the new MST
25         if x != y:
26             if weight < min_cost[v]:
27                 min_cost[v] = weight
28
29             if weight < min_cost[u]:
30                 min_cost[u] = weight
31
32             # Union the sets
33             if rank[x] < rank[y]:
34                 parent[x] = y
35             elif rank[x] > rank[y]:
36                 parent[y] = x
37             else:
38                 parent[x] = y
39                 rank[y] += 1
40
41     return min_cost
42
43 if __name__ == "__main__":
44     n = 6
45     edges = [(0, 1, 2), (0, 2, 4), (0, 3, 1), (1, 2, 3), (1, 4, 5), (2, 3, 2), (2, 5, 7), (3, 5, 1), (3, 6, 3), (4, 5, 6), (4, 6, 4), (5, 6, 5)]
46     deleted_edge = (3, 5)
47     print(new_mst(n, edges, deleted_edge))
48     # Expected output: {0: 0, 1: 2, 2: 2, 3: 3, 4: 5, 5: 2, 6: 3}
```

6. [4 pts] In Dijkstra's algorithm, all distances are initialized to "infinity". Suppose, however, that you cannot represent infinity in your programming language. What actual number could be used in place of "infinity"?

Ans:

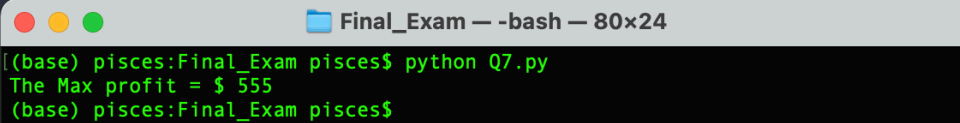
```
val dist[|V|]; //dist from s to v
dijkstra(G,s){
    local PQ = new PQ; //priority Q
    for(i=0;i<|V|;i++){
        dist[i]=inf;
        PQ_ins(PQ,i)
    }
    dist[s] = 0;
    PQ_upd(PQ,s,0);
    while(!empty(PQ)){
        v = PQ_delmin(PQ);
        for(w in adj(v)){
            new = dist[v] + weight[v,w];
            If (new < dist(w) || dist(w) = -1{
                dist[w] = new;
                PQ_upd(PQ,w,new);
            }
        }
    }
}
```

7. [10 pts] You have just won a shopping spree on a game show. You are allowed to fill your shopping cart with everything the cart can hold. The cart has a weight capacity of W . All of the items in the store are marked with both the price and the weight. You are allowed to take multiple items.

7.1. [4 pts] Devise an efficient algorithm to maximize your profit subject to the shopping cart constraints.

My Python code:

```
1 # Hsuan-You Lin Final Exam Question 7
2 import numpy as np
3
4 # weight capacity of the cart
5 W = 20
6
7 def shopping_cart(items):
8     dp = [0 for _ in range(W+1)]
9     for i in range(1, W+1):
10         for (prices, weight) in items:
11             if weight <= i:
12                 dp[i] = max(dp[i], dp[i - weight] + prices)
13
14     return dp[i]
15
16 if __name__ == "__main__":
17     # list of items in the store, each item is a tuple with the price and weight
18     items = [(160, 7), (90, 3), (15, 2)]
19     ans = shopping_cart(items)
20     print("The Max profit = $", ans)
```



Final_Exam — -bash — 80x24

(base) pisces:Final_Exam pisces\$ python Q7.py
The Max profit = \$ 555
(base) pisces:Final_Exam pisces\$

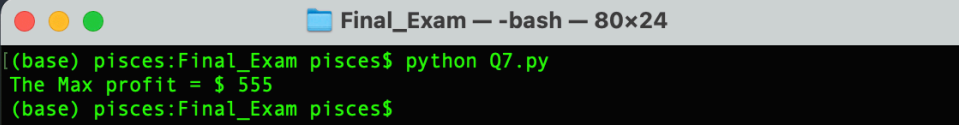
7.2. [2 pts] Apply your algorithm to this problem instance:

W = 20 lbs

Store items: item A: (\$160, 7 lbs). item B: (\$90, 3 lbs), item C:(\$15, 2 lbs)

My Python code:

```
1 # Hsuan-You Lin Final Exam Question 7
2 import numpy as np
3
4 # weight capacity of the cart
5 W = 20
6
7 def shopping_cart(items):
8     dp = [0 for _ in range(W+1)]
9     for i in range(1, W+1):
10         for (prices, weight) in items:
11             if weight <= i:
12                 dp[i] = max(dp[i], dp[i - weight] + prices)
13
14     return dp[i]
15
16 if __name__ == "__main__":
17     # list of items in the store, each item is a tuple with the price and weight
18     items = [(160, 7), (90, 3), (15, 2)]
19     ans = shopping_cart(items)
20     print("The Max profit = $", ans)
```



```
(base) pisces:Final_Exam pisces$ python Q7.py
The Max profit = $ 555
(base) pisces:Final_Exam pisces$
```

7.3. [4 pt] For capacity W, and store item list of length N, give the complexity of your algorithm in terms of both W and N.

Ans: The overall complexity of my algorithm is $O(W*N)$, where N is the number of items in the store and W is the weight capacity of the cart. The inner for loop, which iterates over the items in the store, is executed W times, once for each possible weight in the cart. Since the size of the input (the number of items and the weight capacity of the cart) determines the number of times the inner loop is executed, the complexity of the program is directly proportional to the size of the input.

8. [5 pts] You are given a list of meeting times as pairs of numbers. For example:

#1 (7:30, 9) = from 7:30AM to 9AM

#2 (15, 16:30) = from 3 PM to 4:30 PM

#3 (8, 11) = from 8AM to 11AM

In the list, some meetings will overlap. For example meeting #1 and meeting #3 overlap.

For this problem, you should find the most efficient possible algorithm to merge all overlapping meetings.

Your output should be a list of new meeting times that do not overlap. In the given example, your final output would be:

#1 (7:30, 11) = 7:30 AM to 11 AM, merge #1 & #3 above

#2 (15, 16:30). = 3 PM to 4:30 PM (nothing to merge)

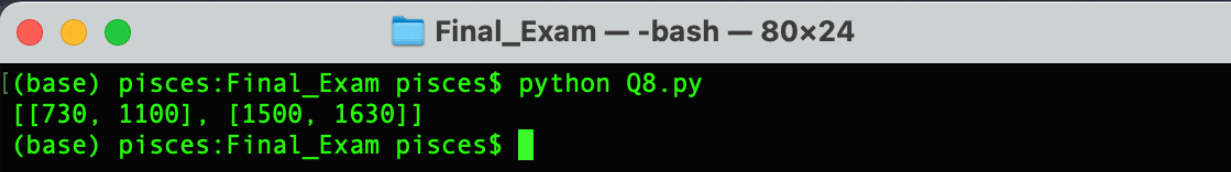
Note that consecutive meetings should be merged. (2, 3) and (3, 4) should be merged into (2,4). Similarly, meetings that are subsumed should be merged into the inclusive meeting. (1, 5) and (2, 4) should be merged into (1, 5).

The input to your merging algorithm will be an array of N pairs of starting and ending times for meetings. Furthermore, meeting start and end times are guaranteed to be on half hour boundaries, between 8:00 AM and 5:30. PM. Finally, the times will be given in military time. 8:00 AM is 800; 5:30 PM is 1730.

In addition to showing your algorithm, analyze the complexity of your algorithm in terms of N (the number of input pairs).

My Python code & explain: The complexity of my algorithm is $O(N \log N)$, where N is the number of meetings in the input list. This is because the initial sorting step has a complexity of $O(N * \log(N))$, and the subsequent iteration through the list of meetings has a complexity of $O(N)$.

```
1  # Hsuan-You Lin Final Exam Question 8.
2  def merge_meetings(meetings):
3      meetings.sort(key=lambda x: x[0])
4
5      merged = []
6      for meeting in meetings:
7          if not merged or merged[-1][1] < meeting[0]:
8              merged.append(meeting)
9          else:
10             merged[-1][1] = max(merged[-1][1], meeting[1])
11
12     return merged
13
14 if __name__ == "__main__":
15     meetings = [[730, 900], [1500, 1630], [800, 1100]]
16
17     print(merge_meetings(meetings)) # should print [(730, 1100), (1500, 1630)]
18
19
```



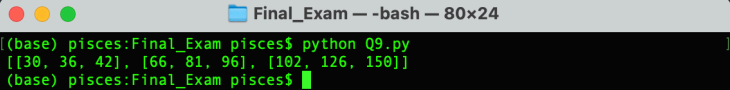
```
(base) pisces:Final_Exam pisces$ python Q8.py
[[730, 1100], [1500, 1630]]
(base) pisces:Final_Exam pisces$
```

9. [5 pts] Write an algorithm that computes nonnegative integer powers of arbitrary 3x3 matrices:

def pow3x3(M,n) = M^n , where M is an arbitrary 3 x 3 matrix.

My Python code & explain: The complexity of this algorithm is $O(N^3)$, where n is the size of the input matrix. This is because the outer loop executes N times, and the inner loop, which computes the matrix multiplication, has a complexity of $O(N^2)$.

```
1 # Hsuan-You Lin Final Exam Question 9.
2 def pow3x3(M, n):
3     # initialize the result to be the identity matrix
4     result = [[1, 0, 0],
5               [0, 1, 0],
6               [0, 0, 1]]
7
8     # iterate n times
9     for i in range(n):
10        # compute the product of the result and M, and update the result
11        result = [[sum(a * b for a, b in zip(result_row, M_col)) for M_col in zip(*M)] for result_row in result]
12
13    return result
14
15 if __name__ == "__main__":
16     # test the pow3x3 function
17     M = [[1, 2, 3],
18          [4, 5, 6],
19          [7, 8, 9]]
20     n = 2
21     print(pow3x3(M, n))
22     # should print [[30, 36, 42], [66, 81, 96], [102, 126, 150]]
```



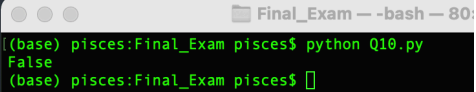
10. [10 pts] A cycle in a directed graph. $G = (V,E)$ is defined as a path $\langle v_1, v_2, \dots, v_1 \rangle$, where the last vertex is the same as the first vertex.

Given a directed graph $G=(V,E)$, determine if G has a directed cycle.

Hint: Use Depth First Search

My python code & explain: The overall time complexity of this program is $O(V+E)$ where V is the number of courses, and E is the number of dependencies.

```
1 # Hsuan-You Lin Final Exam Question 10.
2
3 from collections import defaultdict
4
5 def DFS(courses, request):
6     courseDict = defaultdict(list)
7     checked = [False] * courses
8     path = [False] * courses
9
10    for relation in request:
11        next_course, prev_course = relation[0], relation[1]
12        courseDict[prev_course].append(next_course)
13
14    for curr_course in range(courses):
15        if is_cyclic(curr_course, courseDict, checked, path):
16            return False
17    return True
18
19
20 def is_cyclic(curr_course, courseDict, checked, path):
21     if checked[curr_course]:
22         return False
23     if path[curr_course]:
24         return True
25
26     path[curr_course] = True
27     ret = False
28
29     for child in courseDict[curr_course]:
30         ret = is_cyclic(child, courseDict, checked, path)
31         if ret: break
32
33     path[curr_course] = False
34
35     checked[curr_course] = True
36     return ret
37
38 if __name__ == "__main__":
39     courses = 4
40     request = [(1, 0), (2, 1), (3, 2), (1, 3)]
41     print(DFS(courses, request)) # should print False
42
```



“On my honor, I have neither given nor received any unauthorized aid on this exam.

Start Time: Dec. 10, 2022, P.M 15:00

End Time: Dec. 10, 2022, P.M 18:30