# CS917 Foundations of Computing
# – Algorithms Coursework

2022/2023

## Administration

This assessed piece of coursework must be submitted by **Noon 12:00, Thursday, 24th November 2022**. Submission is done electronically via Tabula, and two files are expected for submission:

1. A PDF of your solutions named Answers.pdf.
2. A file named Practical.py of your python solution for the parts of Question 9.

The solutions submitted must be your **own** work. You are expected to work individually, not in groups. Please show full working where appropriate, detailing how you have arrived at your answer.

## Questions

### Question 1 (10 points)

For each function $f(n)$ and time $t$ in the following table, determine the largest size $n$ (an integer) of a problem that can be solved in time $t$, assuming that the algorithm to solve the problem takes $f(n)$ microseconds. You are free to use Python as a tool to work out answers. **Remember to explain your work progress, not just write down the answers.**

| | 1 second | 1 minute | 1 hour | 1 day | 1 month | 1 year |
|---|---|---|---|---|---|---|
| $n \log_2 n$ | 62746 | 2801417 | 133378058 | 2755147513 | 71870856404 | 797633893349 |
| $n^2$ | 1000 | 7745 | 60000 | 293938 | 1609968 | 5615692 |
| $n^3$ | 100 | 391 | 1532 | 4420 | 13736 | 31593 |
| $2^n$ | 19 | 25 | 31 | 36 | 41 | 44 |
| $n!$ | 9 | 11 | 12 | 13 | 15 | 16 |

*Note*: you can assume there are 30 days in a month and 365 days in a year.

Progress:

1) For ($n \log_2 n$):

We only want to get an integer, so we can use python to help calculate it. The code is like:

```
n = 1
while n * log(n, 2) < 1000000:
    n += 1
result = n - 1
print(result)
```

Based on this, we get

1 second: $n \log_2 n = 1{,}000{,}000 \Longrightarrow n = 62746$

1 minute: $n \log_2 n = 1{,}000{,}000 * 60 \Longrightarrow n = 2801417$

1 hour: $n \log_2 n = 1{,}000{,}000 * 60 * 60 \Longrightarrow n = 133378058$

1 day: $n \log_2 n = 1{,}000{,}000 * 60 * 60 * 24 \Longrightarrow n = 2755147513$

1 month: $n \log_2 n = 1{,}000{,}000 * 60 * 60 * 24 * 30 \Longrightarrow n = 71870856404$

1 year: $n \log_2 n = 1{,}000{,}000 * 60 * 60 * 24 * 30 * 365 \Longrightarrow n = 797633893349$

2）For $n^2$, we can easily calculate it by the root extract function provided by calculator

1 second: $n^2 = 1{,}000{,}000 \Longrightarrow n = 1000$

1 minute: $n^2 = 1{,}000{,}000 * 60 \Longrightarrow n = 7745$

1 hour: $n^2 = 1{,}000{,}000 * 60 * 60 \Longrightarrow n = 60000$

1 day: $n^2 = 1{,}000{,}000 * 60 * 60 * 24 \Longrightarrow n = 293938$

1 month: $n^2 = 1{,}000{,}000 * 60 * 60 * 24 * 30 \Longrightarrow n = 1609968$

1 year: $n^2 = 1{,}000{,}000 * 60 * 60 * 24 * 30 * 365 \Longrightarrow n = 5615692$

3）For $n^3$, same as 2)

1 second: $n^3 = 1{,}000{,}000 \Longrightarrow n = 100$

1 minute: $n^3 = 1{,}000{,}000 * 60 \Longrightarrow n = 391$

1 hour: $n^3 = 1{,}000{,}000 * 60 * 60 \Longrightarrow n = 1532$

1 day: $n^3 = 1{,}000{,}000 * 60 * 60 * 24 \Longrightarrow n = 4420$

1 month: $n^3 = 1{,}000{,}000 * 60 * 60 * 24 * 30 \Longrightarrow n = 13736$

1 year: $n^3 = 1{,}000{,}000 * 60 * 60 * 24 * 30 * 365 \Longrightarrow n = 31593$

3）For $2^n$, we can use math function pow(2, n) to calculate, the code is like:

```
n = 1
while pow(2, n) < 1000000:
    n += 1
result = n - 1
```

1 second: $2^n = 1{,}000{,}000 \Longrightarrow n = 19$

1 minute: $2^n = 1{,}000{,}000 * 60 \Longrightarrow n = 25$

1 hour: $2^n = 1{,}000{,}000 * 60 * 60 \Longrightarrow n = 31$

1 day: $2^n = 1{,}000{,}000 * 60 * 60 * 24 \Longrightarrow n = 36$

1 month: $2^n = 1{,}000{,}000 * 60 * 60 * 24 * 30 \Longrightarrow n = 41$

1 year: $2^n = 1{,}000{,}000 * 60 * 60 * 24 * 30 * 365 \Longrightarrow n = 44$

2）For n!, we can use the factorial(n) provided by python, the code like:

```
n = 1
while factorial(n) < 1000000:
    n += 1
result = n - 1
print(result)
```

1 second: n! = 1,000,000 $\Longrightarrow$ n = 9
1 minute: n! = 1,000,000 * 60 $\Longrightarrow$ n= 11
1 hour: n! = 1,000,000 * 60 * 60 $\Longrightarrow$ n = 12
1 day: n! = 1,000,000 * 60 * 60 * 24 $\Longrightarrow$ n = 13
1 month: n! = 1,000,000 * 60 * 60 * 24*30 $\Longrightarrow$ n = 15
1 year: n! = 1,000,000 * 60 * 60 * 24 *30*365 $\Longrightarrow$ n = 16

## Question 2 (10 points)

Determine whether the following complexity statements are true or false. For each answer, explain your reasoning according to the formal definition of $O$, $\Omega$ and $\Theta$.

(a) $9n^2 + 9$ in $O(n^2)$

(b) $6n^4 - 3n^2 + 3$ in $\Omega(n^4)$

(c) $9n^3 - 7n$ in $O(n^4)$

(d) $n^4 + 2n^2 + 1$ in $\Theta(n^2)$

(e) $n^2 + 6n + 143$ in $\Omega(n)$

(a) True. We can let c = 19 and k=1, then for each n>1, we have $9n^2 + 9 < 19\, n^2$, which satisfies the formal definition of big O.

(b) True. We can let c = 1 and k = 1, then for each n>1, we have $6n^4 - 3n^2 + 3 >= n^4$, which satisfies the formal definition of big Ω.

(c) True. We can let c =9 and k = 1, then for each n>1, we have $9n^3 - 7n < 9\, n^4$, which satisfies the formal definition of big O.

(d) False. We cannot find a $c_1$ and a k to make $n^4 + 2n^2 + 1 <= c_1 * n^2$ holds for each n > k , since the increasing rate of $n^4$ is much larger than $n^2$, once n is linear to positive infinity, $n^4$ will be larger than $c_1 * n^2$ no matter how large c is.

(e) True. We can let c = 1 and k = 1, then for each n >1, we have $n^2 + 6n + 143 >$ n, which satisfies the formal definition of big Ω.

## Question 3 (10 points)

Where possible, apply the master theorem for the following. If not possible, state the reason why.

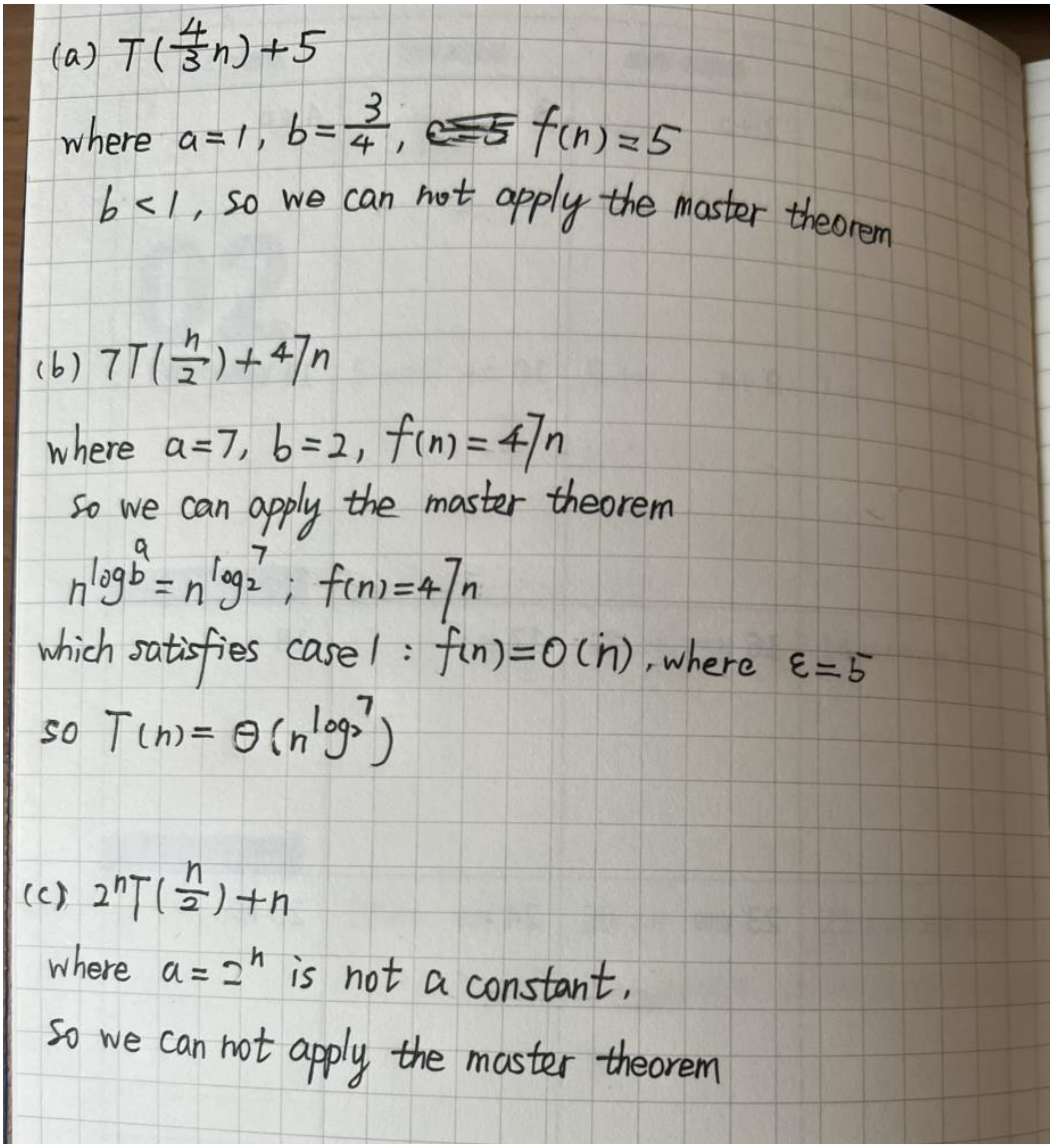
(a) $T(\frac{4n}{3}) + 5$

(b) $7T(\frac{n}{2}) + 47n$

(c) $2^nT(\frac{n}{2}) + n$

(d) $16T(\frac{n}{4}) + 16n^2$

(e) $4T(\frac{5n}{3}) + 50n^3$



(a) $T(\frac{4}{3}n) + 5$

where $a=1$, $b=\frac{3}{4}$, ~~$\varepsilon=5$~~ $f(n)=5$

$b<1$, so we can not apply the master theorem

(b) $7T(\frac{n}{2}) + 47n$

where $a=7$, $b=2$, $f(n)=47n$

So we can apply the master theorem

$n^{\log_b^a} = n^{\log_2^7}$ ; $f(n)=47n$

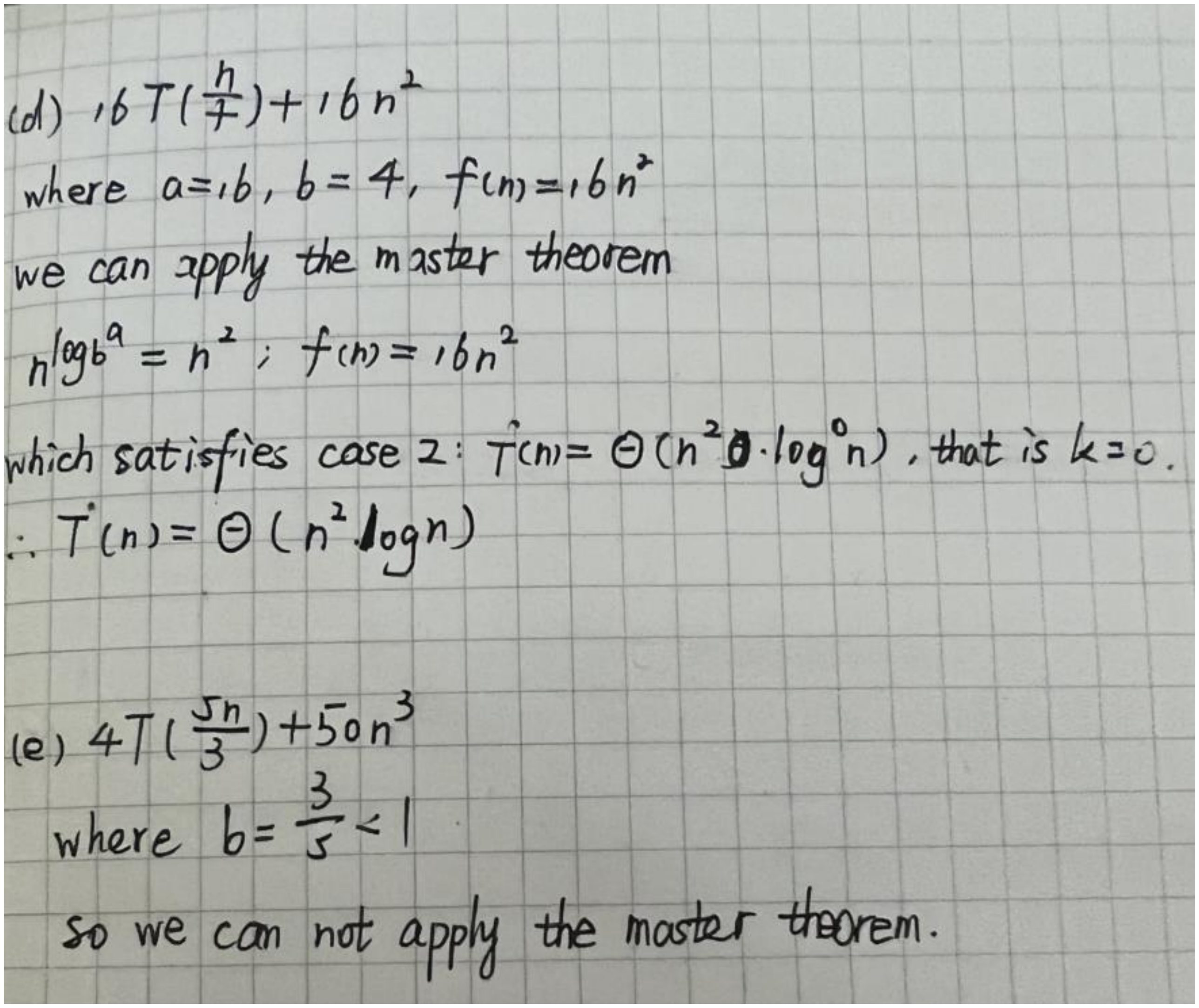
which satisfies case 1 : $f(n)=O(n)$, where $\varepsilon=5$

so $T(n)=\Theta(n^{\log_2^7})$

(c) $2^nT(\frac{n}{2}) + n$

where $a=2^n$ is not a constant,

So we can not apply the master theorem



(d) $16T(\frac{n}{4})+16n^2$

where $a=16$, $b=4$, $f(n)=16n^2$

we can apply the master theorem

$n^{\log_b a} = n^2$ ; $f(n) = 16n^2$

which satisfies case 2: $T(n) = \Theta(n^2 \cdot \log^0 n)$, that is $k=0$.

$\therefore T(n) = \Theta(n^2 \log n)$

(e) $4T(\frac{5n}{3})+50n^3$

where $b = \frac{3}{5} < 1$

so we can not apply the master theorem.

## Question 4 (10 points)

For each of the following input lists, state what you believe to be the most effective sorting algorithm. Justify your reasoning, taking into account complexity, number of operations, memory usage and any other properties of sorted lists. If there are features or attributes of the algorithm that are implementation dependent, state these in your answer.

(a) [1, 2, 3, 4, 6, 5, 7, 10]

(b) [13, 12, 9, 6, 5, 4, 3, 2, 1]

(c) [1, 10, 5, 8, 3, 9, 6, 4, 2]

(d) [6, 5, 6, 5, 9, 11, 1, 23, 7]. Two objects with the same values need to preserve their order after sorting.

(e) [(3, 9), (4, 5), (4, 4), (9, 5), (8, 7), (10, 6)] where each tuple (key, value) is sorted by the key. The ordering of the values must be preserved where the keys are equivalent.

(a) We can use adaptive Bubble Sort for this case.  We can complete the sorting with only (n-1) comparison and one switch operation. It suits the best case of adaptive Bubble Sort, which only costs O(n). Meanwhile, the memory usage is O(1).

(b) We can use Insertion Sort for this case. Since the list has been completely ordered in reverse order. If we use Insertion Sort, we just need to conduct n times insertion and comparison without switch. It suits the best case of Insertion Sort, which costs O(n). The memory usage is O(1).

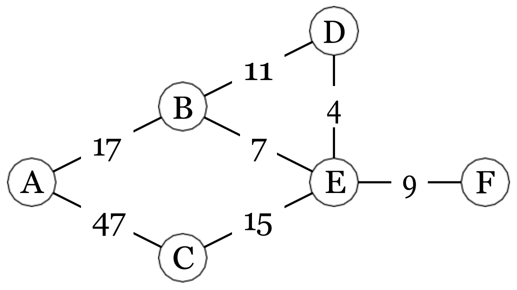
(c) We can use Quick Sort for this case. For the Bubble Sort, Insertion Sort, Selection Sort, it all requires $O(n^2)$ to complete sort, while Quick Sort costs O(nlogn) for this average case. Compared with Merge Sort, Merge Sort requires other computation space O(n) to store its divided lists.

(d) We can use Merge Sort for this case. Since it is a stable sorting algorithm and would not change the initial order of the same values in the list. Compared with the other stable algorithm Bubble Sort and Insertion Sort, its complexity for this case is O(nlogn),which is smaller than the other two.

(e) We can use adaptive Bubble Sort for this case. Since it is a stable sorting algorithm, would not change the initial order of the same keys in the list, and it only requires (n-1) comparison and one switch to complete sort. It suits the best case of adaptive Bubble Sort, which only costs O(n). Meanwhile, the memory usage is O(1).

## Question 5 (5 points)

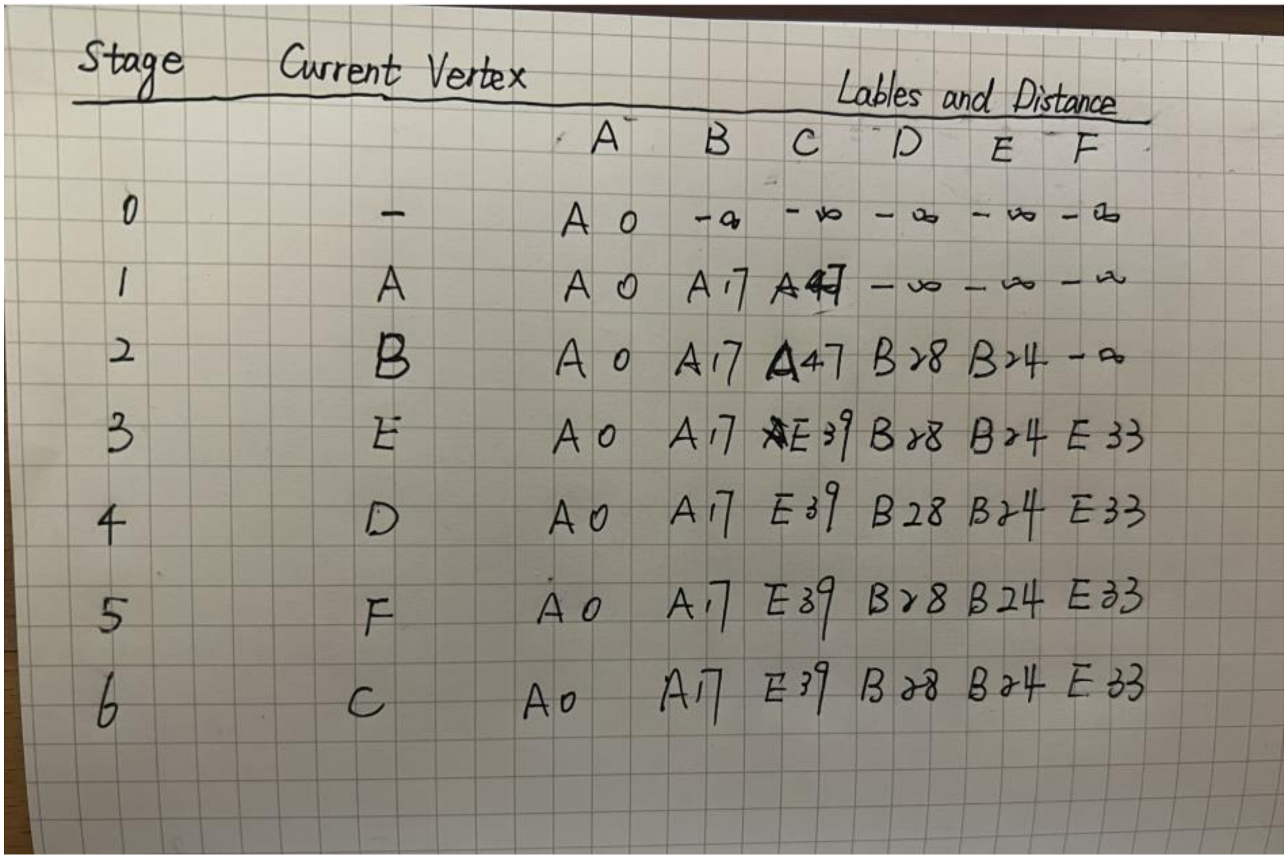
Apply Dijkstra's Algorithm to the following graph, computing the shortest path for all vertices from vertex A.



Present the results in the format of the following table and **write a paragraph** to briefly explain your work process.

| Stage | Current Vertex | Labels and Distances | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | A | | B | | C | | D | | E | | F | |
| 0 | - | A | 0 | - | ∞ | - | ∞ | - | ∞ | - | ∞ | - | ∞ |
| . | . | : | : | : | : | : | : | : | : | : | : | . | . |

Each row states (a) the current stage, (b) the vertex just added to the search graph, and (c) the current updated predecessor node label and distance from A for each vertex in the graph. The vertices should be processed in the order dictated by Dijkstra's Algorithm.



| Stage | Current Vertex | Lables and Distance | | | | | |
|---|---|---|---|---|---|---|---|
| | | A | B | C | D | E | F |
| 0 | – | A 0 | – ∞ | – ∞ | – ∞ | – ∞ | – ∞ |
| 1 | A | A 0 | A 17 | A 47 | – ∞ | – ∞ | – ∞ |
| 2 | B | A 0 | A 17 | A 47 | B 28 | B 24 | – ∞ |
| 3 | E | A 0 | A 17 | ~~A~~E 39 | B 28 | B 24 | E 33 |
| 4 | D | A 0 | A 17 | E 39 | B 28 | B 24 | E 33 |
| 5 | F | A 0 | A 17 | E 39 | B 28 | B 24 | E 33 |
| 6 | C | A 0 | A 17 | E 39 | B 28 | B 24 | E 33 |

**For stage 1, current vertex A can arrive B or C. the distance from A to B is 17, which less than from A to C is 47, so the next vertex is B.**

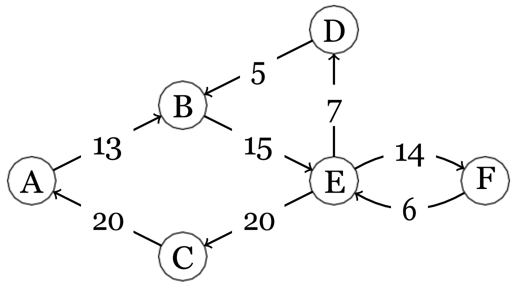
**For stage2, current vertex B can arrive D or E. The total distance from A to C is 47, to D is (17+11) = 28, to E is (17+7) = 24, so the next vertex is E.**

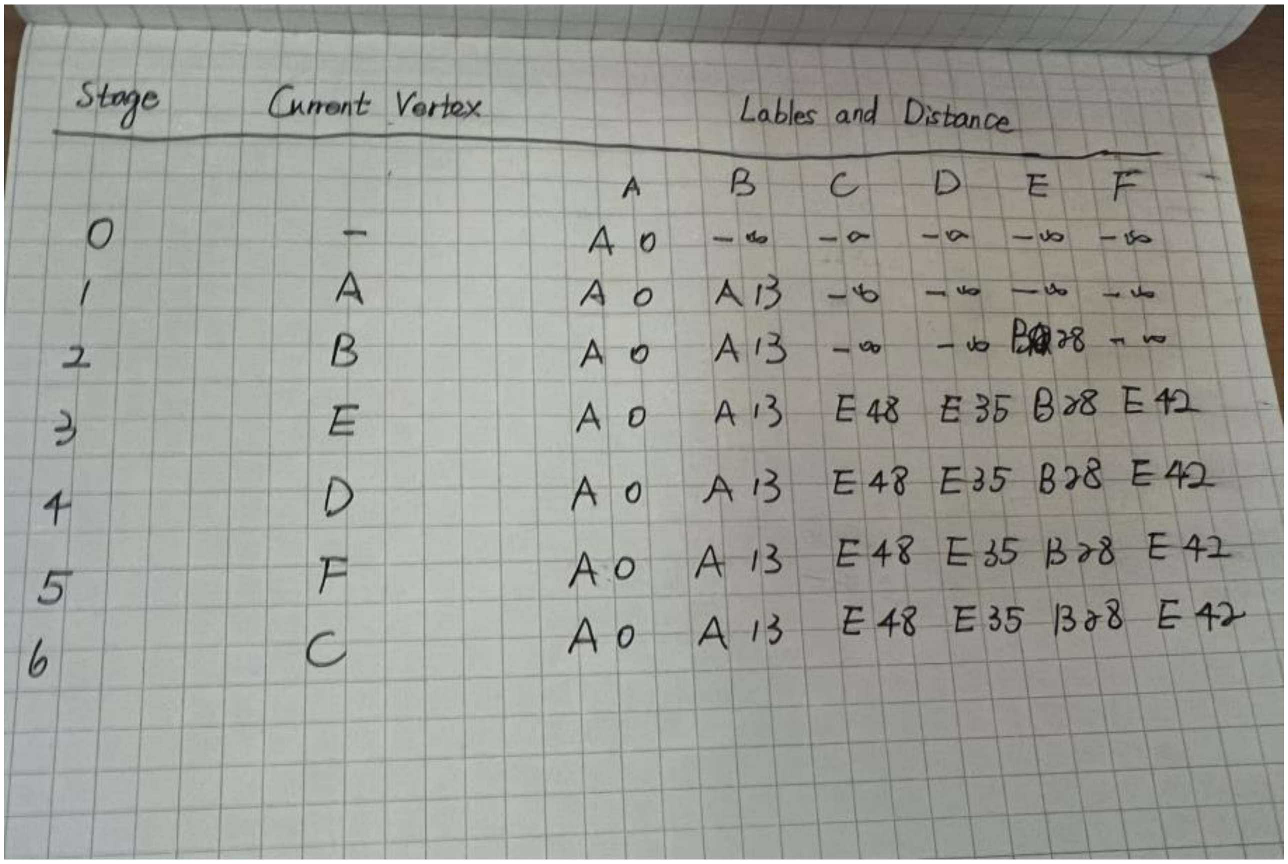
**For stage3, current vertex E can arrive D, F, C.**
**The total distance from A to E then to C is (24+15) = 39 < 47, so we update the total distance from A to C by 39.**
**The total distance from A to E then to D is (24+4) = 28 which equal to A to B to D, we keep it the same.**
**The total distance from A to F is(24+9) = 33.**
**28 < 33 <39**
**So we choose the D as the next vertex**

**For stage4, current vertex D can arrive B or E, but they all have been touched. So we compare 33< 39 and choose F as the next vertex without update.**
**Finally, F can only arrive E, which has been touched, we get C without the distance update**

## Question 6 (5 points)

Apply Dijkstra's Algorithm to the following directed graph, and compute the shortest path for all vertices from vertex A. As in Question 5, present your results in the same table format, and **write a paragraph** to explain your work process.





| Stage | Current Vertex | Lables and Distance | | | | | |
|---|---|---|---|---|---|---|---|
| | | A | B | C | D | E | F |
| 0 | – | A 0 | –∞ | –∞ | –∞ | –∞ | –∞ |
| 1 | A | A 0 | A 13 | –∞ | –∞ | –∞ | –∞ |
| 2 | B | A 0 | A 13 | –∞ | –∞ | B 28 | –∞ |
| 3 | E | A 0 | A 13 | E 48 | E 35 | B 28 | E 42 |
| 4 | D | A 0 | A 13 | E 48 | E 35 | B 28 | E 42 |
| 5 | F | A 0 | A 13 | E 48 | E 35 | B 28 | E 42 |
| 6 | C | A 0 | A 13 | E 48 | E 35 | B 28 | E 42 |

For stage 1, current vertex A can only arrive B(since it is directed graph), so the next vertex is B.

For stage 2, current vertex B can only arrive E, and the total distance from A to E is (13+15) = 28, so the next vertex is E

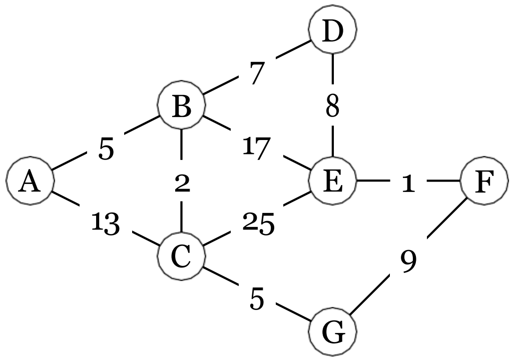
For stage 3, current vertex E can arrive D, C, F, the total distance from A to F is (28+14) = 42, from A to D is (28+7) = 35, from A to C is (28+20) = 48, so the next vertex is D

For stage 4, current vertex D can only arrive B, which has been touched, so we go back to E and compare 42 with 48, so the next vertex is F.

For stage 5, current vertex F can only arrive E, which has been touched, so we go back to E and go to the final node C without updating the total distance.

## Question 7 (10 points)

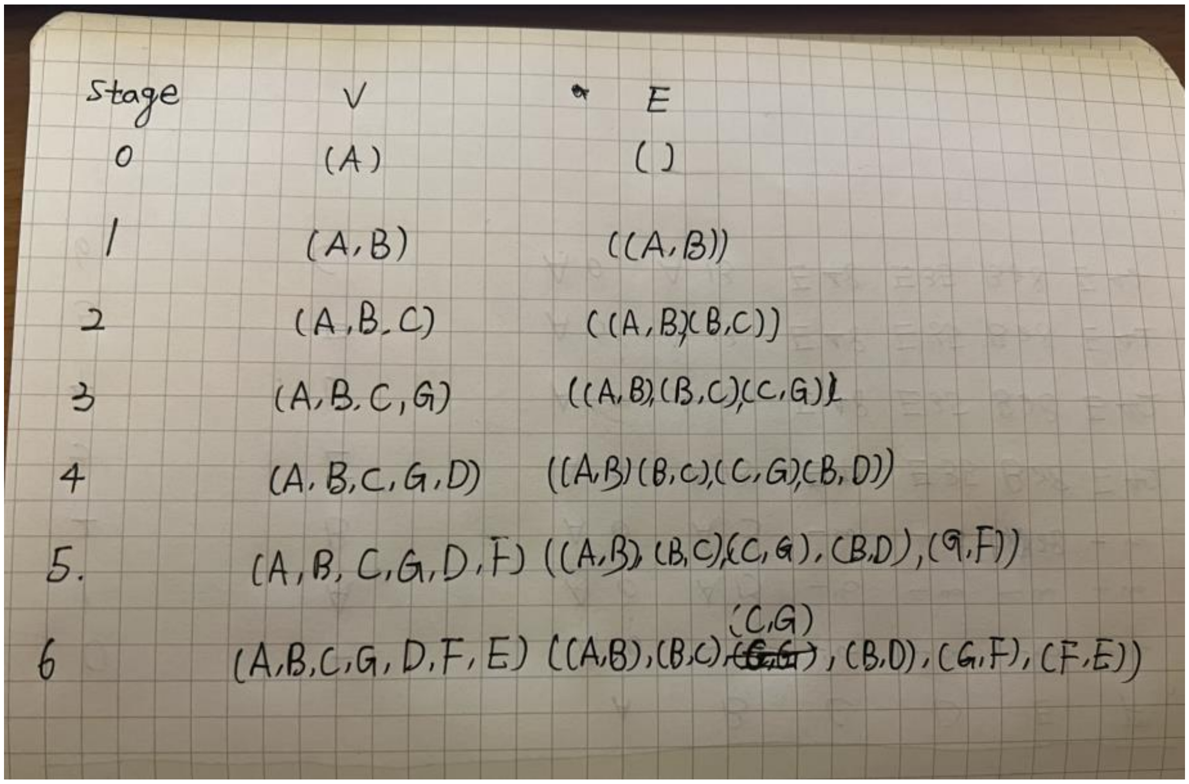
For the following graph:



(a) Apply Prim's Algorithm, beginning at vertex A. Show the results of each stage of the algorithm. In this case, we define a stage as the addition of a new vertex and a new edge to the MST S={V,E}.

Present the results in the format of the following table (note that **values provided are only examples and do not correspond to the solution**), and **write a paragraph** to explain your work process.

| Stage | V | E |
|---|---|---|
| 0 | (A) | () |
| 1 | (A,B) | ((A,B)) |
| : | : | . |
| n | (A,B,C.D,E,F) | ((A,B),(B,C),(C,D),(D,E),(E,F)) |



| Stage | V | E |
|---|---|---|
| 0 | (A) | ( ) |
| 1 | (A,B) | ((A,B)) |
| 2 | (A,B.C) | ((A,B),(B,C)) |
| 3 | (A,B.C,G) | ((A,B),(B,C),(C,G)) |
| 4 | (A.B,C,G,D) | ((A,B)(B,C),(C,G),(B,D)) |
| 5. | (A,B,C,G,D,F) | ((A,B),(B,C),(C,G),(B,D),(G,F)) |
| 6 | (A,B,C,G,D,F,E) | ((A,B),(B,C),(C,G) ~~(G,G)~~,(B,D),(G,F),(F,E)) |

We beginning at vertex A, we can get to B or C,  the distance between A and B is 5,

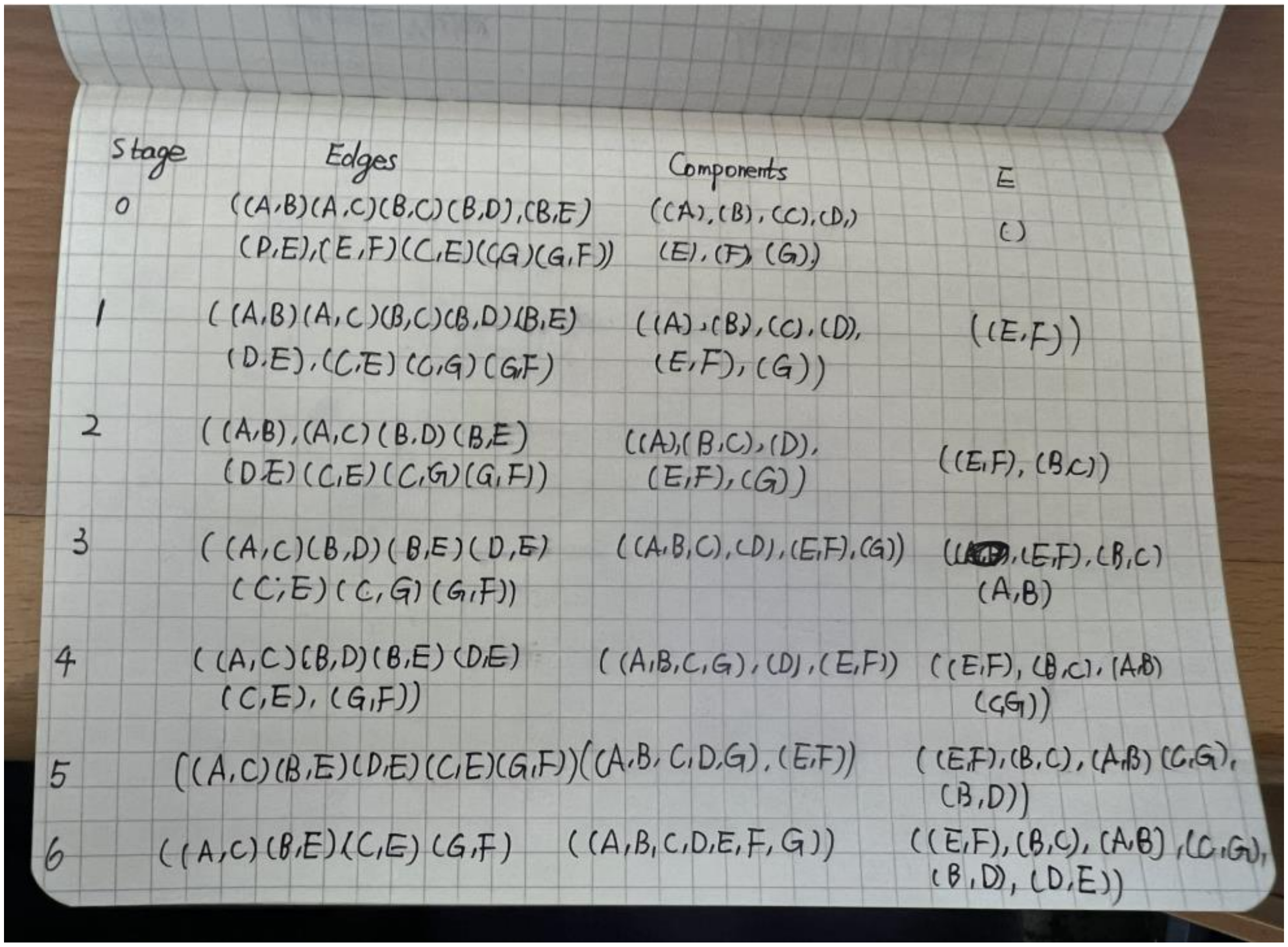
the distance between A  and C is 13 >5, so we add B to the vertex set, and (A,B) to edge set.
For stage1, we can get to C from A, and get to C,D,E from B.
The distance: A to C :13, B to C: 2, B to D:7, B to E: 17
So we add the minimum distance 2, which is C to the vertex set, and (B,C) to edge set.
For stage2, we can get to D,E from B, get to E,G from C
The distance: B to D:7, B to E:17, C to E: 25, C to G:5
So we add the minimum distance 5, which is G to the vertex set, and (C,G) to edge set.
For stage3, we can get to D,E from B, ger to E from C, get to F from G.
The distance: B to D:7, B to E:17, C to E: 25, G to F:9
So we add the minimum distance 7, which is D to the vertex set, and (B,D) to edge set.
For stage4, we can get to E from B, get to E from C, get to F from G
The distance: B to E:17, C to E: 25, G to F:9
So we add the minimum distance 9 ,which is F to the vertex set, and (G,F) to edge set.
Finally, we only leave E , and we can get to E from F with distance equal to 1, so we add
E to the vertex set and (F,E) to the edge set.
We have arrived all the vertex and get the MST.

(b) Apply Kruskal's Algorithm. Show the results of each stage of the algorithm. In this case, we define a stage as the processing of an edge from the graph, whether or not it is added to the MST S=$\{V,E\}$.

Present the results in the format of the following table (**values provided are only examples and do not correspond to the solution**), and **write a paragraph** to explain your work process.

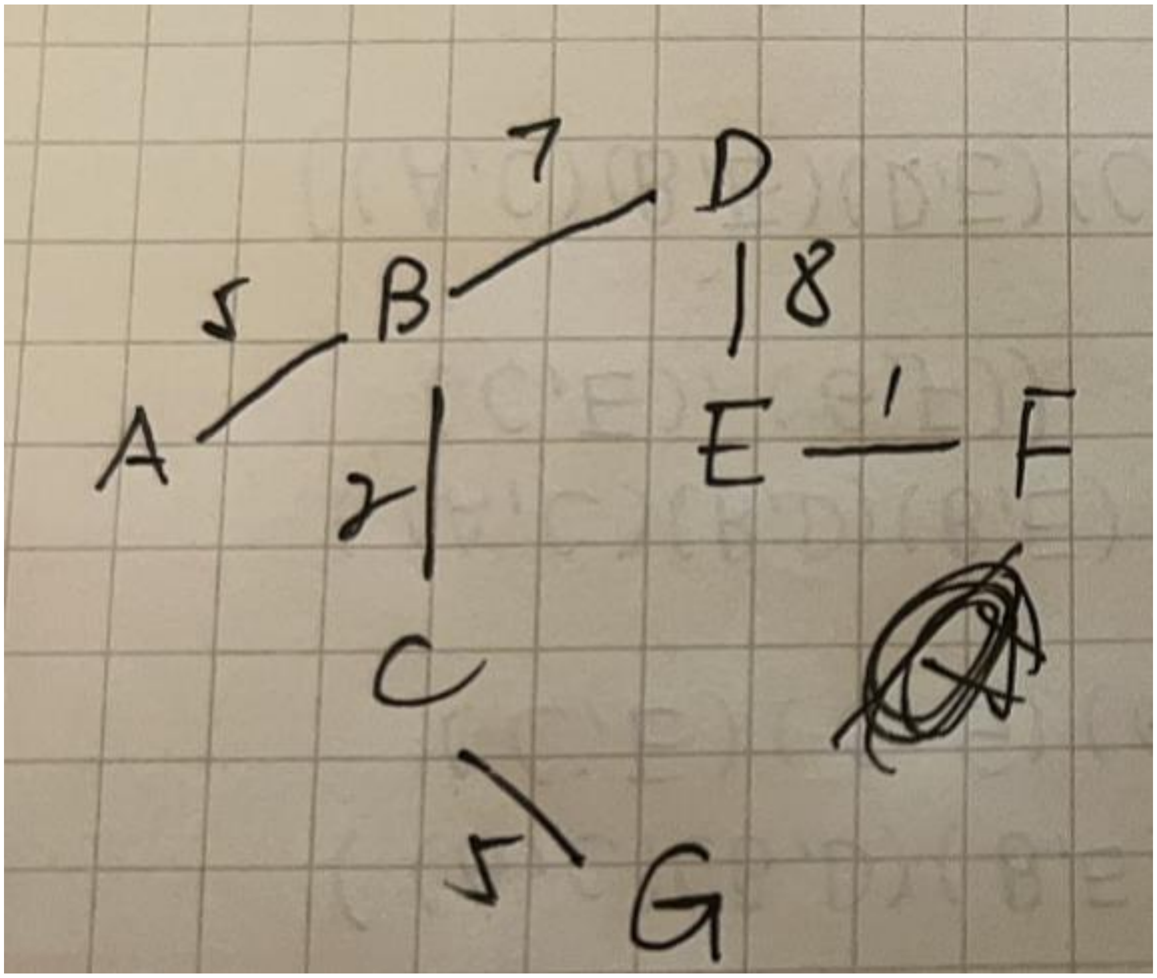
| Stage | Edges | Components | E |
|---|---|---|---|
| 0 | ((A,B), (B,C), (C,D), (D,E), (E,F)) | ((A), (B), (C), (D), (E), (F)) | () |
| 1 | ((B,C), (C,D), (D,E), (E,F)) | ((A,B), (C), (D), (E), (F)) | ((A,B)) |
| . | . | . | . |
| n | | ((A,B,C,D,E,F)) | ((A,B), (B,C), (C,D), (D,E), (E,F)) |

Note the division of the MST Vertices Set into Connected Components.



| Stage | Edges | Components | E |
|---|---|---|---|
| 0 | ((A,B)(A,C)(B,C)(B,D),(B,E) (D,E),(E,F)(C,E)(C,G)(G,F)) | ((A),(B),(C),(D,) (E),(F),(G)) | () |
| 1 | ((A,B)(A,C)(B,C)(B,D)(B,E) (D,E),(C,E)(C,G)(G,F) | ((A),(B),(C),(D), (E,F),(G)) | ((E,F)) |
| 2 | ((A,B),(A,C)(B,D)(B,E) (D,E)(C,E)(C,G)(G,F)) | ((A),(B,C),(D), (E,F),(G)) | ((E,F),(B,C)) |
| 3 | ((A,C)(B,D)(B,E)(D,E) (C;E)(C,G)(G,F)) | ((A,B,C),(D),(E,F),(G)) | ([illegible],(E,F),(B,C) (A,B) |
| 4 | ((A,C)(B,D)(B,E)(D,E) (C,E),(G,F)) | ((A,B,C,G),(D),(E,F)) | ((E,F),(B,C),(A,B) (C,G)) |
| 5 | ((A,C)(B,E)(D,E)(C,E)(G,F)) | ((A,B,C,D,G),(E,F)) | ((E,F),(B,C),(A,B)(C,G), (B,D)) |
| 6 | ((A,C)(B,E)(C,E)(G,F) | ((A,B,C,D,E,F,G)) | ((E,F),(B,C),(A,B),(C,G), (B,D),(D,E)) |

For stage1, we choose the minimum edge (E,F) which is 1 from Edges to E
For stage2, we choose the minimum edge (B,C) which is 2 from Edges to E, and it does not form a cycle.
For stage3, there are two minimum edges (A,B) and (C,G) which are all 5, neither of them would form a cycle, so it does not matter which one is first chose. Here we first choose (A,B) from Edges to E.
Then for stage4, we choose the left one in 3 which is (C,G)
For stage5, we choose the minimum edge (B,D) which is 7 from Edges to E, and it does not form a cycle.
For stage6, we choose the minimum edge (D,E) which is 8 from Edges to E, and it does not form a cycle.
Now we have all the vertex connected in Components, so we get the MST, which is like:



## Question 8 (10 points)

Figure 1 shows left and right rotations in a red-black tree to resolve violations. $T$ denotes the tree, while $x$ and $y$ are two nodes on the tree. The operation LEFT-ROTATE($T$ , $x$)

transforms the configuration of the two nodes on the left into the configuration on the right by changing a constant number of pointers. The inverse operation RIGHT-ROTATE($T$, $y$) transforms the configuration on the right into the configuration on the left. The letters $\alpha$, $\beta$, and $\gamma$ represent arbitrary sub-trees.

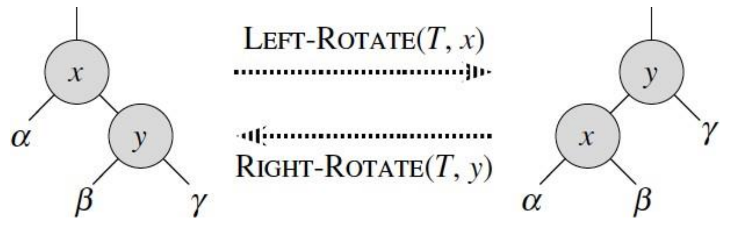


Figure 1: The rotation operations on a red-black tree

The pseudo-code for LEFT-ROTATE is shown in Algorithm 1. A node's left child, right child and parent are represented as .left, .right and .p respectively. T.nil represents an empty node. This algorithm assumes that x.right ≠ T.nil and that the root's parent is T.nil. Complete the pseudo-code for RIGHT-ROTATE in Algorithm 2.

**Algorithm 1** LEFT-ROTATE(T, x)

```
Require: x.right ≠ T.nil, T.root.p == T.nil
 1: y = x.right                      // set y
 2: x.right = y.left                 // turn y's left substree into x's right subtree
 3: if y.left ≠ T.nil then
 4:     y.left.p = x
 5: end if
 6: y.p = x.p                        // link x's parent to y
 7: if x.p == T.nil then
 8:     T.root = y
 9: else if x == x.p.left then
10:     x.p.left = y                 // link y to be the left child of x's parent
11: else
12:     x.p.right = y
13: end if
14: y.left = x                       // put x on y's left
15: x.p = y
```

**Algorithm 2** RIGHT-ROTATE(T, y)

```
Require: y.left ≠ T.nil, T.root.p == T.nil
 1: x = y.left                          // set x
 2: y.left = x.right
 3: if x.right != T.nil then
 4:     x.right.p = y
 5: end if
 6: x.p = y.p                           // link y's parent to x
 7: if y.p == T.nil then
 8:     T.root = x
 9: else if y == y.p.right then
10:     y.p.right = x                   // link x to be the left child of y's parent
11: else
12:     y.p.left = x
13: end if
14: x.right = y                         // put y on x's right
15: y.p = x
```

## Question 9 (30 points)

The following parts are intended to test your application of algorithms and understanding of their performance. Implement your solutions within the file Practical.py and include it in your submission. You will be assessed not only on functionality but also on your approach and implementation, so include comments detailing how your code functions and apply appropriate good programming practices. You may write additional methods or classes for any extra data structures you feel you may need to answer the question, but **do not change the signatures of existing functions in the skeleton file Practical.py** (we will check your program based on testing these functions). An example of the usage of these tasks is provided in the skeleton file. When it is requested that you explain something specifically for a question, you may include the answer in Answers.pdf.

(a) **Morse Code**

You've taken up a summer job at a lighthouse, when you notice a light blinking in the distance - it looks like morse code! With your computer close at hand, you decide to write a method that can decode morse code as quickly as possible. Morse code is a series of dot (.) and dash (-) symbols that when combined in a certain order represent a single letter. This ordering is fixed, as found in Figure 2. By processing each letter, we can construct a word.

In the file Practical.py, write a method called morseDecode that will take a single input of a list of Strings. Each string in the list will represent a single letter of a word in morse code (i.e. a string of . and - ). Your method should take the list of strings and return a string that contains the message translated into English. Detail your design decisions and implementation of this method in Answers.pdf.

(b) **Incomplete Morse Code**

Unfortunately, the seas are very busy this night. Passing ships have been blocking your view of the light. Luckily, we still are able to figure out how many characters there were per morse code letter, but we are missing the first character of every morse-code letter.

In Practical.py, write a method called partialMorseDecode that is passed a single word in morse code as a list of strings (each index being a single letter in morse code, the same as part (a)). To represent a missing symbol, we will use the lowercase letter 'x'. The function should return a list of strings, consisting of valid words only. We consider a valid word to be any word that exists in a provided file, dictionary.txt. You may write any additional data structures or methods that you believe you will need. It is recommended to begin with that you use short words. As part of your answer, discuss your implementation approach and the complexity of this algorithm in Answers.pdf.

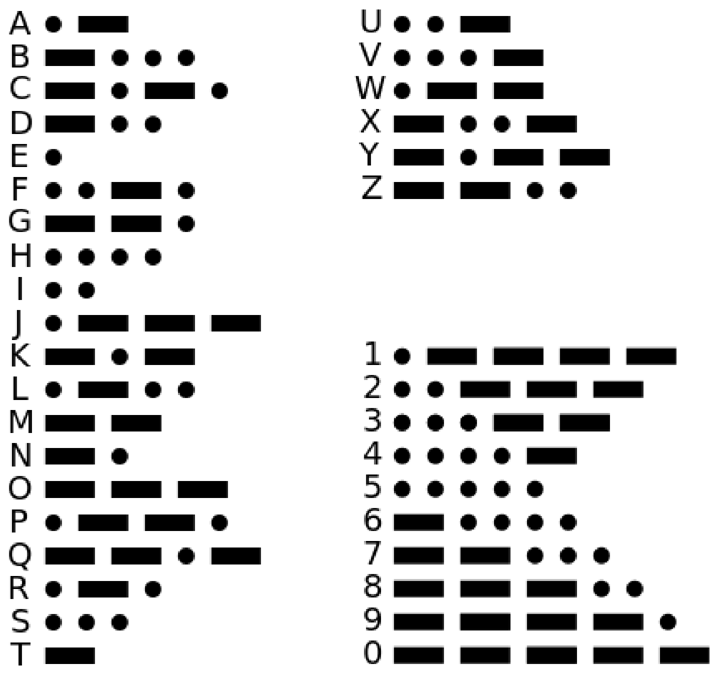


Figure 2: Morse Code

(c) **The Maze**

The morse code has led you on a search for treasure! Following the message's instructions, you find yourself at the top of a set of stairs, with a maze lying below. You can see which parts are walls and which parts are open, but the maze is winding and a path through difficult to spot. A mystery person calls out, mapping the maze from above, dividing it into a grid like system as in Figure 3:

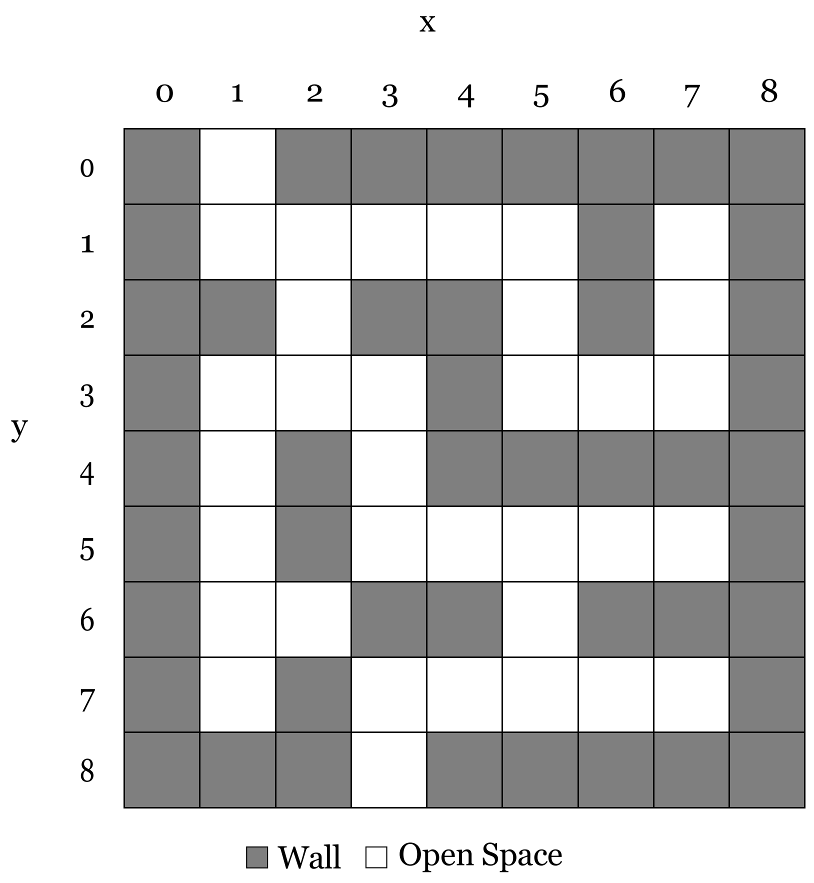


Figure 3: Maze

The above is an example maze, but the maze you are looking at is far more complex. Assuming a coordinate system that begins with (0,0) at the top left, you must store information about the state of the maze, and use that information to map a route through it. *You may assume all mazes used for testing are rectangles, i.e.,* $M \times N$ *where* $M$ *and* $N$ *are determined by the maximum of the given* $x$ *and* $y$ *coordinates respectively*. Given a class called Maze, complete these three methods:

- addCoordinate(x, y, blockType): The x represents the x coordinate on the grid. The y represents the y coordinate on the grid. The blockType can be either 0, to represent an open space, or 1 to represent a wall. The only coordinates you know about in a maze are those added by addCoordinate. If you have been told nothing about a coordinate, but it fits within the current height and width of the maze (i.e. the highest x and y values that have been passed), you may assume it is a wall.
- printMaze(): This method should print a representation of the maze, using a star (*) character for walls, and a empty space for open spaces, similar to how Figure 2 is presented (but with symbols instead of squares). You should go up to the maximum height and width of the coordinates you know of so far from addCoordinate.

- findRoute(x1, y1, x2, y2): This method returns a list of (x, y) tuples that map a route to follow, starting from (x1, y1) and moving to (x2, y2). They should be ordered in the list such that you can always move from the coordinate at index i to the coordinates at index i+1, starting at (x1, y1) in index 0 and finishing at coordinates (x2,y2) in the last index position. If no route is available, return an empty list.

In Answers.pdf detail your implementation and why you chose that approach. Again, you may write additional methods and classes if you wish, as long as they are in the Practical.py file and the methods described above are completed.

(a) First, we should create a dictionary to store the morse code, where value is the real alphabet or number and key is the corresponding morse code.
Then we can loop the whole list and convert each item in the list to the real letter according to the dictionary we create and add it to the final resulting string.
For the algorithm complexity, it requires n times operation of check, n times of adding, which is O(n).

(b) Since we lost the first letter in morse code, it means each morse code has two possibilities. We can use Itertools.product to get all the possible combinations of the letters and check whether they are in the dictionary.
Here I firstly use Itertools.product to get all the possible combinations, and to get the letters in the tuples, I design a recursion algorithm GetLetter(newtuple), where the base case is the algorithm has got the first element in the tuple and return the final result. The recursion case is that we get the second element of the current tuple and then add it to the end of the result string, return GetLetter(newtuple[0]) + result.
Finally, we traversal all the combinations to check whether the final result is in the dictionary and return the result list.
The algorithm complexity is the complexity of the recursion algorithm, to calculate it, we should use the Master Theorem. The expression of my algorithm can be written as

$$T(n) = \begin{cases} O(1) & n = 1 \\ T\left(\frac{n}{2}\right) + 1 & n \geq 2 \end{cases}$$

Applying the Master Theorem case 2, we get the complexity is Θ(logn). Since we need n times loop outside the recursion algorithm, the total complexity is Θ(nlogn).

(c) To store all the required information in the Maze class, I use two integers x and y to store the width and height of the maze, and use an array list blocktype[y_position][x_position] to store the position and the value is 0 or

1(represents space or wall). More specific, if the coordinate is (2,1), and it is a wall, then blocktype[1][2] = 1.
To implement addcoordinate function, we should create a new arraylist newarray[][] based on the width and height, and set all the values to 1 (so the coordinate which are not added will automatically become wall). Then we should copy the values from the blocktype to newarray, and update the blocktype using newarray.
To print maze, it only requires two loops in blocktype, which prints all the lines and rows.

To search a best path, here I chose BFS algorithm. Firstly, we should create a new matrix maze which has the same width and height of the blocktype. While all the values in the new maze would set be 0.
Based on the given start point(x1,y1), we set the values of maze[y1][x1] to be 1. Then we use BFS to search all the points where the values of maze is 0 and the values of blocktype is 0 too(it means it is a space and has not been touched yet.) Each turn we add 1 to a counter k, and update the values of the maze[y][x] where x and y satisfies the requirement stated above, until the end point is reached. In this BFS algorithm, finally we find all the points can be reached and set the values of them as the order they are reached.
After BFS algorithm, we get a maze with all the possible reached points whose value set as the order they are reached. Then we only need to find the neighbors of each point from the end point, and we add the point to the start of the final list.
BFS is one of the best and simplest algorithm to solve this maze problem, with BFS, we can directly make use of the data structure of Maze itself. We firstly search all the possible points in the maze, and using K neighbours to find the final path.