Assignment2

Exercise 1

(a) $f(n) = 11 * n$
$n = \frac{932*60*60}{11} = 305018$

(b) $f(n) = 7 * n^2$
$n = \sqrt{\frac{932*60*60}{7}} = 692$

(c) $f(n) = 17 * \frac{\sqrt{n}}{5}$
$n = \left(\frac{5*932*60*60}{17}\right)^2 = 973820678200$

(d) $f(n) = \frac{1560870*\log_2 n}{11}$
$n = 2^{\frac{11*932*60*60}{1560879}} = 13120094$

(e) $f(n) = 7^{n-2}$
$n = 2 + log_7(932 * 60 * 60) = 9$

```python
class Task1:
    hour = 3600
    operations = 932
    def complexity_a(self):
        return math.trunc(self.operations*self.hour/11)
    def complexity_b(self):
        return math.trunc(math.sqrt(self.operations*self.hour/7))
    def complexity_c(self):
        return math.trunc(25*((self.operations*self.hour/17)**2))
    def complexity_d(self):
        return math.trunc(2**(self.operations*self.hour*11/1560870))
    def complexity_e(self):
        return math.trunc(math.log(self.operations*self.hour,7)+2)
```

```
305018
692
973820678200
13120094
9
```

Exercise 2

(a) $9 * n^2 + 9$ in $O(n^2)$
True.
$9 * n^2 + 9 < 10 * n^2$ for $n > 3$, $9n^2 \sim O(n^2)$

(b) $6 * n^4 - 3 * n^2 + 3$ in $\Omega(n^4)$
True.
$6 * n^4 - 3 * n^2 + 3 >= n^4$ for every n

(c) $9 * n^3 - 7 * n$ in $O(n^4)$
True.
$9 * n^3 - 7 * n < n^4$ for $n >= 9$, $n^4 \sim O(n^4)$

(d) $n^4 + 2 * n^2 + 1$ in $\Theta(n^2)$
False.
It is impossible to find any k and c1, c2 to satisfy
$n^4 + 2 * n^2 + 1 <= c1 * n^2$
$n^4 + 2 * n^2 + 1 >= c2 * n^2$
for $n > k$

(e) $n^2 + 6 * n + 143$ in $\Omega(n)$
True.
$n^2 + 6 * n + 143 >= n$ for every n

Exercise 3

(a) $T(\frac{4*n}{3}) + 5$
It is impossible to apply the theorem. b should be greater than 1.

(b) $7 * T(\frac{n}{2}) + 47 * n$
$f(n)$ can be expressed as $O(n^k * (logn)^p)$
$a = 7, b = 2, x = log_2 7, k = 1$
$k < x => T(n) = \Theta(n^{log_2 7})$

(c) $2^n * T(\frac{n}{2}) + n$
It is impossible to apply the theorem. a should be constant and should not be a function of n.

(d) $16 * T(\frac{n}{4}) + 16 * n^2$
$f(n)$ can be expressed as $O(n^k * (logn)^p)$
$a = 16, b = 4, x = log_4 16 = 2, k = 2, p = 0$
$T(n) = \Theta(n^2 * logn)$

(e) $4 * T(\frac{5*n}{3}) + 50 * n^3$
It is impossible to apply the theorem. b should be greater than 1.

Exercise 4

(a) Insertion sort and Adaptive Bubble Sort. The reason in choosing insertion sort is that this sorting algorithm is adaptive and for each element k in slots 1..n, first check will be whether array[k] ¿= array[k-1] and if it is true then go to the next element. For our input list it is required to make comparison several times and then insert element 5 before element 6. Also an adaptive bubble sort is a good solution for this particular case where the number of unsorted elements is very small (=1).

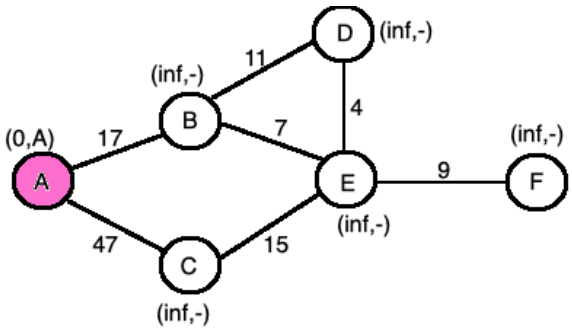
(b) Quick sort. The input list is reversed. The pivot needs to be picked from the middle of the input list, so the number of elements on each side of the pivot would be half with every iteration time, which is good case for quicksort. In addition to that, quicksort is in-place which makes it better compared to merge sort.

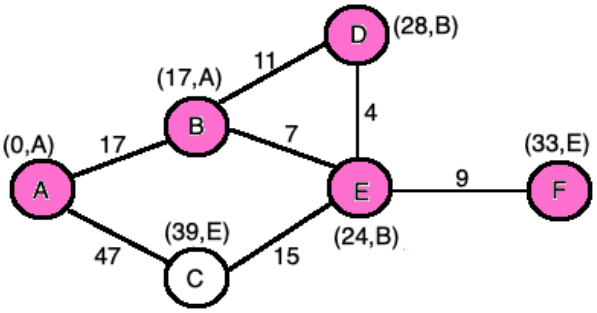
(c) Quick sort. The pivot needs to be picked from the middle of the input list. Here it is element 3. We have 2 elements which are greater than 3 in the left side and 2 elements which are smaller than 3 in the right side. The time complexity is $O(n * logn)$. Quick sort is also better than merge sort in terms of memory usage.

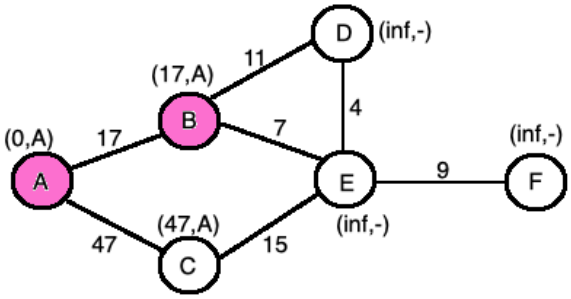
(d) Merge sort. The complexity is $O(n*logn)$. This sorting algorithm is stable so objects with the same values keep their original order.

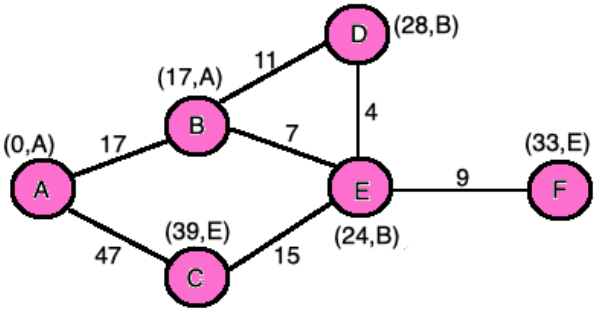
(e) Insertion sort. As we can see there is only one unsorted pair, so similar to part a of this exercise we can use insertion sort.

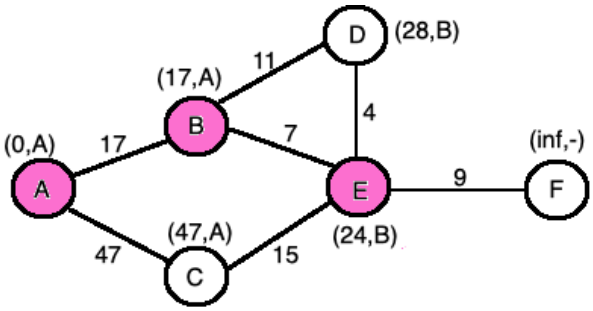
Exercise 5

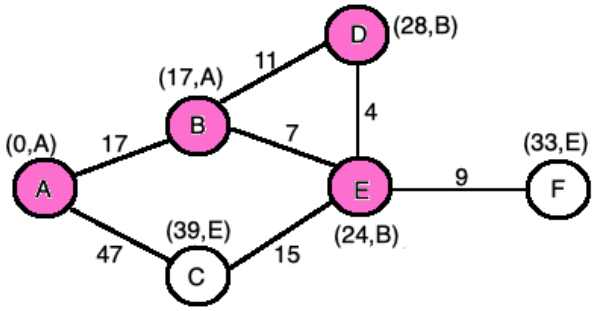

D
(inf,-)
11
(inf,-)
B
4
(0,A)
17
7
(inf,-)
A
E
9
F
47
15
(inf,-)
C
(inf,-)


D
(28,B)
11
(17,A)
B
4
(0,A)
17
7
(33,E)
A
E
9
F
(39,E)
47
15
(24,B)
C

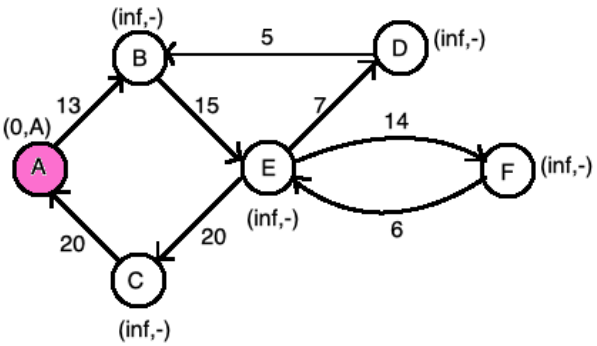

D
(inf,-)
11
(17,A)
B
4
(0,A)
17
7
(inf,-)
A
E
9
F
(47,A)
47
15
(inf,-)
C

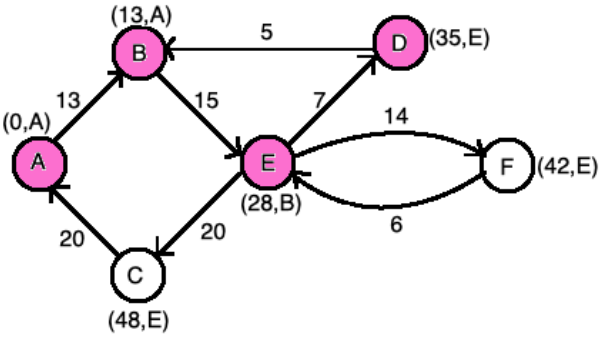

D
(28,B)
11
(17,A)
B
4
(0,A)
17
7
(33,E)
A
E
9
F
(39,E)
47
15
(24,B)
C

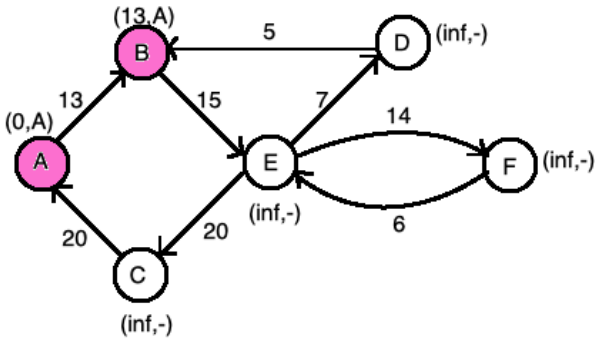

D
(28,B)
11
(17,A)
B
4
(0,A)
17
7
(inf,-)
A
E
9
F
(47,A)
47
15
(24,B)
C

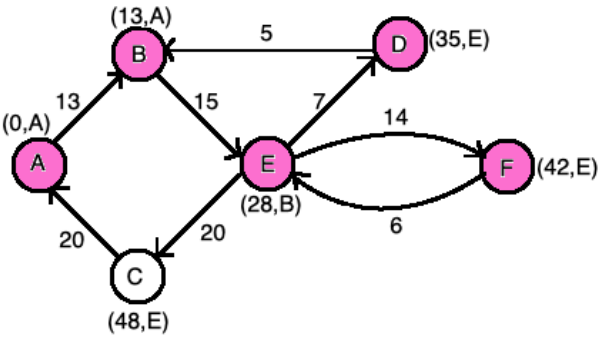

D
(28,B)
11
(17,A)
B
4
(0,A)
17
7
(33,E)
A
E
9
F
(39,E)
47
15
(24,B)
C

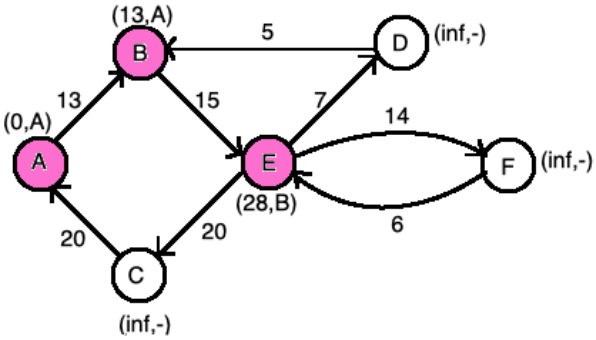
Exercise 6

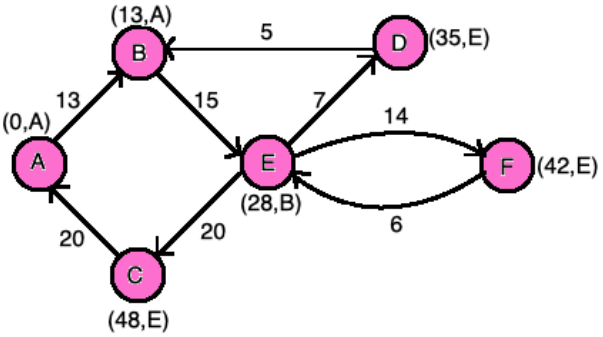

(inf,-)
B
5
D
(inf,-)
13
15
7
14
(0,A)
A
E
F
(inf,-)
(inf,-)
6
20
20
C
(inf,-)


(13,A)
B
5
D
(35,E)
13
15
7
14
(0,A)
A
E
F
(42,E)
(28,B)
6
20
20
C
(48,E)


(13,A)
B
5
D
(inf,-)
13
15
7
14
(0,A)
A
E
F
(inf,-)
(inf,-)
6
20
20
C
(inf,-)


(13,A)
B
5
D
(35,E)
13
15
7
14
(0,A)
A
E
F
(42,E)
(28,B)
6
20
20
C
(48,E)


(13,A)
B
5
D
(inf,-)
13
15
7
14
(0,A)
A
E
F
(inf,-)
(28,B)
6
20
20
C
(inf,-)


(13,A)
B
5
D
(35,E)
13
15
7
14
(0,A)
A
E
F
(42,E)
(28,B)
6
20
20
C
(48,E)

Exercise 7

Prim's Algorithm

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,E)	((A,B),(B,C),(C,G),(B,D),(D,E))
6	(A,B,C,G,D,E,F)	((A,B),(B,C),(C,G),(B,D),(D,E),(E,F))

Kruskal's Algorithm

Stage	*Edges*	*Components*	*E*
0	((A,B),(A,C),(B,C),(B,D),(B,E),(D,E),(C,E),(E,F),(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))	((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,G,D),(E,F))	((E,F),(B,C),(A,B),(C,G),(B,D))
6	((A,C),(B,E),(C,E),(G,F))	((A,B,C,G,D,E,F))	((E,F),(B,C),(A,B),(C,G),(B,D),(D,E))

Exercise 8

Algorithm 1 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.left then
10:     y.p.left = x                  ▷ link x to be the left child of y's parent
11: else
12:     y.p.right = x
13: end if
14: x.right = y                                         ▷ put y on x's right
15: y.p = x
```

Exercise 9

(a) In this part, I have created a dictionary of morse codes. Keys are the morse codes consisted from symbols '.' or '-'. Values are corresponding letters or numbers.

Function morseDecode receives a list of morse code as an input and decode it into english word using created dictionary and getting the value from its key.

(b) My approach in this task includes using recursion in order to find the whole list of possible words that could be created from the input data. In each iteration i of recursion one more sign "-" or "." is added to the whole set of previous words of length $i-1$. This way the size of the list of words doubles every time.

We have n recursive steps, for each letter in the word. For the i'th step, we will iterate through every element of word (to subsequently add the two prefixes for the new set of words). At the i'th step, the length of the list of words is $2^{(i-1)}$. Thus, the total number of processing steps is given by: $1 + 2 + \ldots + 2^{(n-1)} = 2^n - 1$. We also do decode, but every time we decode only two letters. Hence the final complexity could be estimated as $O(2^n)$.

(c) In this task I have chosen Breadth First Search Algorithm. Breadth First Search Algorithm is optimal and is guaranteed to find the best solution that exists. It starts at some given cell of the maze and explores the neighbouring cells first, before moving to the next level neighbours. For storing the cells which will be exploring a queue is used. Also, it is necessary to store all of the visited cells in order to escape loop. Time complexity for(worst case is $O(|V| + |E|)$, where $|V|$ is a number of cells and $|E|$ is a number of edges.

In this particular case in python, we may use "deque" (double-ended queue). First of all, we put the initial cell into the queue and create a set of visited cells (at this step it consists only of the initial cell). Then repeat the following procedure until reaching the final cell or visit all available cells.

- Take the first node to visit from the queue
- Check if the destination is reached. If True, then it is the end of algorithm and we can return the created path
- Else put all possible neighbours (not visited, not wall, exist) at the end of the queue and at the set of visited cells
- Repeat while there are nodes to visit