Laboratory Assignment 2

You can use “random,” “string,” “time,” and “heapq” libraries. Please do not use any other library. All unspecified library lines will be suppressed and if your code fails, your submission will be graded “0” points

Problem 1 – Generate an array of random entries [4 pts]

Write a function that takes four `integer` inputs (`n,wmin,wmax,tflag`) and returns a one dimensional list with random entries.

- Function parameter names do not matter, but the function should expect 4 parameters.
- First parameter (`n`) defines the size of the list.
- Fourth parameter (`tflag`) defines the entry type (i.e. `string`, `integer`, `float`).
- Second and third inputs (`wmin,wmax`) inclusively determines
  - the minimum and maximum length of the string, if the requested type is `string`,
  - the minimum and maximum values, if the requested type is `integer` or `float`.
- If the requested type of the entries is string, then the elements of the list should consist of alphanumeric characters:
  `"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"`
- You can use `random` and `string` libraries for this problem.
- `inputs :`
  - `n ∈ ℤ and n ≥ 1`
  - `tflag = { tflag ∈ {0,1,2}: 0=string (default), 1=integer, 2=float}`
  - `(wmin,wmax) ∈ ℤ and wmax > wmin ≥ 1: default values wmin=1, wmax=10`
- `output : x = { x : x is a list}`

```
>>> rl = problem1(10,2,10,0)
>>> print(rl)
['c5OR2NW', '6mRfkx8ke', 'x8HNGv', 'Q1V82i', 'n5J', 'kLjmBBGtZ', 'nCk', 'Y4J',
'QLWRfRF', 'r4XjZSb']
>>> rl = problem1(5,2,4)
>>> print(rl)
['nhd', 'Plw', 'ded', '2jJ', 'L6A']
>>> rl = problem1(5)
>>> print(rl)
['DB55k8YZo4', 'hT2', 'cvbJ7veYqW', 'SrCITLJ3', 'RP3rVeY']
>>> rl = problem1(10,2,100,1)
>>> print(rl)
[32, 42, 15, 49, 83, 23, 58, 15, 82, 18]
>>> rl = problem1(10,2,10,2)
```

```
>>> print(rl)
[9.08105060553546, 2.459769101547682, 7.617757176888987, 7.454574283370493,
8.059754433994314, 2.2215160332625654, 7.832767956664679, 6.077978179671733,
6.600619510784702, 7.400605079882618]
>>> rl = problem1(5,2,3)
>>> print(rl)
['R9A', 'br', '9cJ', 'qHG', 'rw']
```

Explanation: rl has 5 elements. Elements are string, since the default value of tflag=0. Length of strings are 2 and 3, since wmax=3 and wmin=2. Note that len(rl[i]) ∈ [wmin=2, wmax=3] inclusive.

```
>>> rl = problem1(5,2,3,1)
>>> print(rl)
[3, 3, 3, 3, 3]
>>> rl = problem1(5,2,3,1)
>>> print(rl)
[3, 3, 2, 2, 3]
```

Problem 2 – Insertion Sort by pair wise swaps [4 pts]

Write a function that takes a list with string/integer/float elements and sorts the list using insertion sort by pair wise comparisons and swaps. The function measures and returns the CPU time in nanoseconds as an `integer`.

- Function parameter names do not matter, but the function should expect 1 parameter.
- You can use `time` library, specifically measure the CPU time using `perf_counter_ns()`[1]. You can simply call this function at the first and at the last lines of the function, then calculate the difference and return.
- Note that for this function,
  - sorted list should not be returned, instead use the mutable property of the list,
  - implement the swaps of elements as explained in the lecture.
- `inputs :`
  - `rl = { rl : rl is a list}`
- `output : t = { t : t is an integer}`

```
>>> rl=['c5OR2NW', '6mRfkx8ke', 'x8HNGv', 'Q1V82i', 'n5J', 'kLjmBBGtZ', 'nCk',
'Y4J', 'QLWRfRF', 'r4XjZSb']
>>> t=problem2(rl)
>>> print(rl)
['6mRfkx8ke', 'Q1V82i', 'QLWRfRF', 'Y4J', 'c5OR2NW', 'kLjmBBGtZ', 'n5J', 'nCk',
'r4XjZSb', 'x8HNGv']
>>> print(t)
17900
>>> rl=[32, 42, 15, 49, 83, 23, 58, 15, 82, 18]
>>> t=problem2(rl)
```

```
>>> print(rl)
[15, 15, 18, 23, 32, 42, 49, 58, 82, 83]
>>> print(t)
18100
>>> rl=[9.08105060553546, 2.459769101547682, 7.617757176888987,
7.454574283370493, 8.059754433994314, 2.2215160332625654, 7.832767956664679,
6.077978179671733, 6.600619510784702, 7.400605079882618]
>>> t=problem2(rl)
>>> print(rl)
[2.2215160332625654, 2.459769101547682, 6.077978179671733, 6.600619510784702,
7.400605079882618, 7.454574283370493, 7.617757176888987, 7.832767956664679,
8.059754433994314, 9.08105060553546]
>>> print(t)
22800
```

Problem 3 – Insertion Sort by pair wise comp. & pop/insert `[3 pts]`

Write a function that takes a list with string/integer/float elements and sorts the list using insertion sort by first determines the position of the element by pair wise comparisons, then uses the pop and insert for the insertion of the associated element, instead of swaps. The function measures and returns the CPU time in nanoseconds as an `integer`.

- Function parameter names do not matter, but the function should expect 1 parameter.
- You can use `time` library, specifically measure the CPU time using `perf_counter_ns()`[1]. You can simply call this function at the first and at the last lines of the function, then calculate the difference and return.
- Note that for this function,
  - sorted list should not be returned, instead use the mutable property of the list,
  - implement the sorting such that first the index of the list for the insertion is determined by pair wise comparisons, then `[].pop()` and `[].insert()` are used for insertion of elements.
- `inputs :`
  - `rl = { rl : rl is a list}`
- `output : t = { t : t is an integer}`

```
>>> rl=['c5OR2NW', '6mRfkx8ke', 'x8HNGv', 'Q1V82i', 'n5J', 'kLjmBBGtZ', 'nCk',
'Y4J', 'QLWRfRF', 'r4XjZSb']
>>> t=problem3(rl)
>>> print(rl)
['6mRfkx8ke', 'Q1V82i', 'QLWRfRF', 'Y4J', 'c5OR2NW', 'kLjmBBGtZ', 'n5J', 'nCk',
'r4XjZSb', 'x8HNGv']
>>> print(t)
16500
>>> rl=[32, 42, 15, 49, 83, 23, 58, 15, 82, 18]
>>> t=problem3(rl)
>>> print(rl)
[15, 15, 18, 23, 32, 42, 49, 58, 82, 83]
```

```
>>> print(t)
16000
>>> rl=[9.08105060553546, 2.459769101547682, 7.617757176888987,
7.454574283370493, 8.059754433994314, 2.2215160332625654, 7.832767956664679,
6.077978179671733, 6.600619510784702, 7.400605079882618]
>>> t=problem3(rl)
>>> print(rl)
[2.2215160332625654, 2.459769101547682, 6.077978179671733, 6.600619510784702,
7.400605079882618, 7.454574283370493, 7.617757176888987, 7.832767956664679,
8.059754433994314, 9.08105060553546]
>>> print(t)
39300
```

Problem 4 – Insertion Sort by binary search `[6 pts]`

Write a function that takes a list with string/integer/float elements and sorts the list using insertion sort by first determines the position of the element by binary search, then inserts the associated element. The function measures and returns the CPU time in nanoseconds as an `integer`.

- Function parameter names do not matter, but the function should expect 1 parameter.
- You can use `time` library, specifically measure the CPU time using `perf_counter_ns()`[1]. You can simply call this function at the first and at the last lines of the function, then calculate the difference and return.
- Note that for this function,
  - sorted list should not be returned, instead use the mutable property of the list,
  - implement the sorting such that first the index of the list for the insertion is determined by binary search as explained in the lecture, then insert the associated element to the determined index.
- `inputs :`
  - `rl = { rl : rl is a list}`
- `output : t = { t : t is an integer}`

```
>>> rl=['c5OR2NW', '6mRfkx8ke', 'x8HNGv', 'Q1V82i', 'n5J', 'kLjmBBGtZ', 'nCk',
'Y4J', 'QLWRfRF', 'r4XjZSb']
>>> t=problem4(rl)
>>> print(rl)
['6mRfkx8ke', 'Q1V82i', 'QLWRfRF', 'Y4J', 'c5OR2NW', 'kLjmBBGtZ', 'n5J', 'nCk',
'r4XjZSb', 'x8HNGv']
>>> print(t)
57100
>>> rl=[32, 42, 15, 49, 83, 23, 58, 15, 82, 18]
>>> t=problem4(rl)
>>> print(rl)
[15, 15, 18, 23, 32, 42, 49, 58, 82, 83]
>>> print(t)
22600
```

```
>>> rl=[9.08105060553546, 2.459769101547682, 7.617757176888987,
7.454574283370493, 8.059754433994314, 2.2215160332625654, 7.832767956664679,
6.077978179671733, 6.600619510784702, 7.400605079882618]
>>> t=problem4(rl)
>>> print(rl)
[2.2215160332625654, 2.459769101547682, 6.077978179671733, 6.600619510784702,
7.400605079882618, 7.454574283370493, 7.617757176888987, 7.832767956664679,
8.059754433994314, 9.08105060553546]
>>> print(t)
24400
```

Problem 5 – Merge Sort `[3 pts]`

Write a function that takes a list with string/integer/float elements and sorts the list using merge sort. The function measures and returns the CPU time in nanoseconds as an `integer`.

- Function parameter names do not matter, but the function should expect 1 parameter.
- You can use `time` library, specifically measure the CPU time using `perf_counter_ns()`[1]. You can simply call this function at the first and at the last lines of the function, then calculate the difference and return.
- Note that for this function,
  - sorted list should not be returned, instead use the mutable property of the list,
  - you can use the sort function of Python [].sort(), since the Python uses merge sort for sufficiently large inputs.
- `inputs :`
  - `rl = { rl : rl is a list}`
- `output : t = { t : t is an integer}`

```
>>> rl=['c5OR2NW', '6mRfkx8ke', 'x8HNGv', 'Q1V82i', 'n5J', 'kLjmBBGtZ', 'nCk',
'Y4J', 'QLWRfRF', 'r4XjZSb']
>>> t=problem5(rl)
>>> print(rl)
['6mRfkx8ke', 'Q1V82i', 'QLWRfRF', 'Y4J', 'c5OR2NW', 'kLjmBBGtZ', 'n5J', 'nCk',
'r4XjZSb', 'x8HNGv']
>>> print(t)
1600
>>> rl=[32, 42, 15, 49, 83, 23, 58, 15, 82, 18]
>>> t=problem5(rl)
>>> print(rl)
[15, 15, 18, 23, 32, 42, 49, 58, 82, 83]
>>> print(t)
2900
>>> rl=[9.08105060553546, 2.459769101547682, 7.617757176888987,
7.454574283370493, 8.059754433994314, 2.2215160332625654, 7.832767956664679,
6.077978179671733, 6.600619510784702, 7.400605079882618]
>>> t=problem5(rl)
>>> print(rl)
```

```
[2.2215160332625654, 2.459769101547682, 6.077978179671733, 6.600619510784702,
7.400605079882618, 7.454574283370493, 7.617757176888987, 7.832767956664679,
8.059754433994314, 9.08105060553546]
>>> print(t)
3000
```

Problem 6 – Heap Sort `[5 pts]`

Write a function that takes a list with string/integer/float elements and sorts the list using heap sort. The function measures and returns the sorted list and the CPU time in nanoseconds as an `integer`.

- Function parameter names do not matter, but the function should expect 1 parameter.
- You can use `time` library, specifically measure the CPU time using `perf_counter_ns()`[1]. You can simply call this function at the first and at the last lines of the function, then calculate the difference and return.
- You can use `heapq` library for heap operations[2].
- Note that for this function, unlike the other sorting problems, sorted list **should be returned**.
- `inputs :`
  - `rl = { rl : rl is a list}`
- `outputs :`
  - `rl = { rl : rl is the sorted list}`
  - `t = { t : t is an integer}`

```
>>> rl=['c5OR2NW', '6mRfkx8ke', 'x8HNGv', 'Q1V82i', 'n5J', 'kLjmBBGtZ', 'nCk',
'Y4J', 'QLWRfRF', 'r4XjZSb']
>>> rl,t=problem6(rl)
>>> print(rl)
['6mRfkx8ke', 'Q1V82i', 'QLWRfRF', 'Y4J', 'c5OR2NW', 'kLjmBBGtZ', 'n5J', 'nCk',
'r4XjZSb', 'x8HNGv']
>>> print(t)
15200
>>> rl=[32, 42, 15, 49, 83, 23, 58, 15, 82, 18]
>>> rl,t=problem6(rl)
>>> print(rl)
[15, 15, 18, 23, 32, 42, 49, 58, 82, 83]
>>> print(t)
12700
>>> rl=[9.08105060553546, 2.459769101547682, 7.617757176888987,
7.454574283370493, 8.059754433994314, 2.2215160332625654, 7.832767956664679,
6.077978179671733, 6.600619510784702, 7.400605079882618]
>>> rl,t=problem6(rl)
>>> print(rl)
[2.2215160332625654, 2.459769101547682, 6.077978179671733, 6.600619510784702,
7.400605079882618, 7.454574283370493, 7.617757176888987, 7.832767956664679,
8.059754433994314, 9.08105060553546]
```

```
>>> print(t)
13800
```

Notes:

[1]https://docs.python.org/3/library/time.html
[2]https://docs.python.org/3/library/heapq.html