Programming Assignment #2

1 Rules

This programming assignment is about heap. In addition, we want to give you an experience of solving a practice problem and testing your programs. Testing the codes is an essential and critical skill for a software developer.

1. Implement the data structures on your own. Modern programming languages already include basic data structures; for instance, in C++, you can directly create a doubly linked list using `std::list`. However, this assignment is not about utilizing these pre-existing interfaces. Your implementations should be based on simple data types, such as primitive data types and pointers. The goal is to implement the data structures and algorithms as learned in class.
2. Write comments and use meaningful naming conventions. Commenting on your code and using variable names that reflect their functionality are critical for software engineering, as they make the code understandable. The rule of thumb is that others should feel comfortable and find your code tractable when reading it. Points may be deducted if the code is difficult to understand.
3. Use your favorite programming language. You are permitted to use your preferred programming language, which includes, but is not limited to, C, C++, and Python. However, ensure you adhere to the guidelines stated in point (1).
4. Test your program thoroughly. We will specify several required tasks to test your data structures. Test your program independently and include the results in your report.
5. **No late submission.** This assignment is closed to the end of semester and please submit your report/source codes on time.

2 Part 1: Implementing a Min Heap

2.1 Introduction

In this part of the assignment, you are tasked with implementing a Min Heap data structure. A Min Heap is a complete binary tree where the value at each node is less than or equal to the values at its children nodes. This property ensures that the smallest element is always at the root of the tree. Min Heaps are crucial in various applications, such as priority queues, scheduling algorithms, and efficient sorting.

2.2 Task Description

You are required to implement a Min Heap in the programming language of your choice. The key operations you need to support are:

- **Insertion:** Add a new element to the heap while maintaining the Min Heap property.
- **Extraction:** Remove the smallest element (the root) from the heap and restructure the heap to maintain the Min Heap property.
- **Peek:** Return the smallest element from the heap without removing it.

Your implementation should be efficient in terms of both time and space complexity.

2.3 Testing Suggestions

Thorough testing is crucial to ensure the correctness of your Min Heap implementation. Here are some suggestions for testing your Min Heap:

1. **Empty Heap:** Test operations like peek and remove on an empty heap to ensure they handle edge cases properly.
2. **Random Inserts:** Insert a series of random elements into the heap and verify that the smallest element is always at the root.
3. **Sequential Inserts and Removals:** Insert elements in a specific order, then remove them one by one, ensuring the Min Heap property is maintained throughout.

Remember to include edge cases in your testing, such as inserting duplicate elements or handling negative numbers. Document your testing strategy, the tests you performed, and the outcomes to ensure your Min Heap implementation is robust and reliable.

3 Part 2: Implementing Heap Sort

3.1 Introduction

In this part of the assignment, you are required to implement the Heap Sort algorithm. Heap Sort is an efficient sorting algorithm that uses a heap data structure to sort elements. It involves building a heap from the input data and then repeatedly removing the largest (or smallest) element from the heap and adding it to the end of the sorted array. This part will test your understanding and ability to apply the Min Heap structure you implemented in Part 1.

3.2 Task Description

You are to implement the Heap Sort algorithm using the Min Heap data structure. The algorithm should follow these steps:

- **Build a Min Heap:** Convert the input array into a Min Heap.
- **Extract Elements:** Repeatedly remove the smallest element from the heap (the root) and move it to the current position in the array, reducing the heap size each time and maintaining the Min Heap property.
- **Repeat:** Continue removing the smallest element from the heap and placing it in the array until all elements are sorted.

Your implementation should efficiently sort the array in ascending order and work for arrays of all sizes.

3.3 Testing Suggestions

To ensure the reliability and efficiency of your Heap Sort implementation, consider the following testing strategies:

1. **Random Arrays:** Test your Heap Sort on arrays filled with random integers. Verify that the output is a sorted version of the input.
2. **Edge Cases:** Test arrays with edge cases such as empty arrays, arrays with all identical elements, and arrays with negative numbers.
3. **Large Arrays:** Assess the performance and efficiency of your Heap Sort by sorting large arrays and measuring the time taken.
4. **Reverse Order:** Test your algorithm on arrays that are sorted in reverse order to ensure it can handle worst-case scenarios.

Document your approach to testing, the variety of tests performed, and any issues uncovered during the process. This will demonstrate the thoroughness of your testing and the reliability of your implementation.

4 Part 3: Using Min-Heap to Find the K-th Minimum Difference

Now, apply your heap implementation to solve a practical problem.

Problem: Imagine a floor scattered with playing cards, each card displaying a number. Our task is to find two cards whose numerical difference is the smallest. However, we aim not merely for the smallest but for the k-th smallest difference. For instance, if the cards on the floor are 8, 2, 6, 5, and 2, and we seek the second smallest difference, our answer would be 1, since the smallest difference is 0.

Definition: In this scenario, you are provided with an array of integers, C, representing the cards scattered on the floor, each identified by an integer. Your objective is to write a program that accepts a number k and the array C, and computes the k-th smallest absolute difference between any two cards in the array.

Your Program: You should employ your heap to fulfill this task. Proceed as follows:

- Construct a Min Heap from the absolute differences between all pairs of distinct cards in the array.
- Execute k removals (extract-min operations) from the Min Heap.
- Return the value obtained after the k-th removal.

If the problem is unclear, refer to the example codes provided. They read from an `input.txt` file, which contains several lines, each representing a problem instance where the first number is k, followed by the array C (cards). These examples calculate the answers and write them into an output file. However, note that this is merely illustrative; solutions not utilizing a heap, as shown in these examples, will be considered incorrect and awarded *zero* points.

The following is NOT the solution! Below are reference codes (`Python3`) demonstrating how to find the k-th minimum difference (using Python's default sort), which should not be replicated for this assignment:

exec.py:

```
#!/usr/bin/env python3

import sys

def readFile(filename):
   "Read input file"
   linesList = list(open(filename, 'r').read().splitlines())
   instances = []
   for i in range(len(linesList)):
      instance_str = linesList[i].split(' ')
      instance_int = [ int(instance_str[j]) for j in range(len(instance_str)) ]
      instances.append(instance_int)
   return instances


def check(instance):
   if len(instance) < 3:
      return False
```

```
    return True

def solve(instance):
    if not check(instance):
        return -1

    k = instance[0]
    cards = instance[1:]

    diff = []
    for i in range(len(cards)):
        for j in range(i + 1, len(cards)):
            if i == j:
                continue
            else:
                diff.append(abs(cards[i] - cards[j]))

    diff.sort()
    return diff[k - 1]


def writeResult(filename, solutions):
    "Write output file"
    file = open(filename, 'w', encoding = 'UTF-8')
    for i in range(len(solutions)):
        if i > 0:
            file.write('\r\n')
        file.write(str(solutions[i]))
    file.close()


if __name__ == '__main__':

    # Command line
    if len(sys.argv) == 1:
        infilename = 'input.txt'
        outfilename = 'output.txt'
    elif len(sys.argv) == 3:
        infilename = sys.argv[1]
        outfilename = sys.argv[2]
    else:
        print("Usage: python3 exec.py [<inputfile> <outputfile>]")
        assert False

    # Read input file
    instances = readFile(infilename)

    # Execute
    solutions = [ solve(instance) for instance in instances ]

    # Output result
    writeResult(outfilename, solutions)
```

input.txt:

```
3 1 5 2 3
17 4 5 2 3 13 7 8
21 4 5 11 10 13 2 9
```

output.txt:

```
2
6
11
```

4.1 Testing Guide

Your next task is to design a series of test cases to validate your own program. Please write about your implementation and the tests in your report.

4.1.1 Testing Strategies

1. **Random Problem Generation:** Design scripts to randomly generate a series of problem instances, including arrays of numbers with different lengths and ranges. Ensure to include edge cases, such as very large or very small numbers.
2. **Negative Numbers:** Create test cases that include negative numbers in the array to ensure your program correctly calculates absolute differences.
3. **Multiple Same Differences:** Test scenarios where the array contains elements that lead to multiple identical differences. This will help verify that your program can handle cases with repeated differences correctly.
4. **Duplicate Numbers in Input:** Ensure your test cases include arrays with duplicate numbers to check how your program deals with such instances.

4.1.2 Comparing with Reference Codes

After running your tests, compare the outputs of your program with the outputs generated by the reference code. This comparison should be done automatically by your testing script to avoid manual errors and to facilitate bulk testing.

4.1.3 Reporting

In your report, detail the following:

- The testing strategies you employed.
- How you compared your results with the reference codes.
- The number of tests you conducted.
- Examples of test cases, including edge cases.
- Any bugs identified during the testing process and how they were (or could be) resolved.

Remember, thorough testing is crucial for developing reliable software. By following these guidelines, you should be able to identify and fix bugs, thereby improving the robustness of your solution.

5 What to submit

You need to submit a `.zip` file for source codes and a `.pdf` file for a report.

5.1 Source codes:

Please wrap all your source codes into a `.zip` file and submit it to the class.

5.2 Report:

In your report, you are expected to provide a comprehensive explanation of your source code and implementation details for all three parts. Describe the logic behind your approach, the data structures used, and any challenges you encountered during the implementation process.

Additionally, you should detail your testing strategy, including the different types of test cases you designed and the results of these tests. Remember, thorough testing and clear, informative comments within your code are crucial aspects of software development. These practices not only help in maintaining and understanding the code but also in identifying and fixing potential bugs.

The completeness and depth of your report, including both the explanation of your code and the documentation of your testing, will be considered during grading. Ensure that your report is well-organized, clear, and reflects the effort and understanding you have put into this assignment.