CS101 Algorithms and Data Structures
Fall 2023
Homework 3

Due date: 23:59, October 29th, 2023

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. `CamScanner` is recommended.
5. When submitting, match your solutions to the problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.

1. (6 points) Sorting practice

Given an array:

$$1, 4, 7, 3, 5, 2, 8, 6$$

(a) (2') Run Insertion Sort for this array. Write down the array **after each** outer iteration.

```
for(int k = 1; k < n; k++){
    for(int j = k; j > 0; j--){
        if( array[j - 1] > array[j] )
           swap(array[j - 1], array[j]);
        else
            break;
    }
    print(array);
}
```

Solution:

(i) $\underline{1\ 4}\ 7\ 3\ 5\ 2\ 8\ 6$

(ii) $\underline{1\ 4\ 7}\ 3\ 5\ 2\ 8\ 6$

(iii) $\underline{1\ 3\ 4\ 7}\ 5\ 2\ 8\ 6$

(iv) $\underline{1\ 3\ 4\ 5\ 7}\ 2\ 8\ 6$

(v) $\underline{1\ 2\ 3\ 4\ 5\ 7}\ 8\ 6$

(vi) $\underline{1\ 2\ 3\ 4\ 5\ 7\ 8}\ 6$

(vii) $\underline{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}$

(b) (2') Run Flagged Bubble Sort for this array. Write down the array **after each** outer iteration.

```
for(int i = n-1; i > 0; i--){
    int max_t = array[0];
    bool sorted = true;
    for(int j = 1; j <= i; j++){
        if(array[j] < max_t){
            array[j-1] = array[j];
            sorted = false;
        }
        else{
             array[j-1] = max_t;
             max_t = array[j];
        }
    }
    array[i] = max_t;
```

```
    print(array);
    if(sorted) break;
}
```

Solution:

(i) 1 4 3 5 2 7 6 8

(ii) 1 3 4 2 5 6 7 8

(iii) 1 3 2 4 5 6 7 8

(iv) 1 2 3 4 5 6 7 8

(v) 1 2 3 4 5 6 7 8

(c) (2') Run Merge Sort for this array. Write down the array **after each** merge and underline the sub-array being merged.

Solution:

(i) $\underline{1}\ \underline{4}\ \underline{7}\ \underline{3}\ \underline{5}\ \underline{2}\ \underline{8}\ \underline{6}$

(ii) $\underline{1\ 4}\ \underline{3\ 7}\ \underline{2\ 5}\ \underline{6\ 8}$

(iii) $\underline{1\ 3\ 4\ 7}\ \underline{2\ 5\ 6\ 8}$

(iv) $\underline{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}$

2. (6 points) Which Sort?

Given a sequence

$$A = \langle 5, 4, 8, 7, 6, 2, 1, 3, 7, 9 \rangle,$$

we have performed different sorting algorithms and printed the intermediate results. Note that the steps below are **not** necessarily consecutive steps in the algorithm, but they are guaranteed to be in the correct order.

For each group of steps, guess ($\surd$) what the algorithm is. The algorithm might be one among the following choices:

- Insertion-sort, implemented in the way that avoids swapping elements
- Flagged Bubble-sort
- Merge-sort

(a) (2')

$$\langle 4, 5, 7, 8, 6, 2, 1, 3, 7, 9 \rangle,$$
$$\langle 2, 4, 5, 6, 7, 8, 1, 3, 7, 9 \rangle,$$
$$\langle 1, 2, 3, 4, 5, 6, 7, 8, 7, 9 \rangle.$$

$\surd$ **Insertion-sort** ◯ Bubble-sort ◯ Merge-sort

(b) (2')

$$\langle 4, 5, 6, 7, 8, 2, 1, 3, 7, 9 \rangle,$$
$$\langle 4, 5, 6, 7, 8, 1, 2, 3, 7, 9 \rangle,$$
$$\langle 1, 2, 3, 4, 5, 6, 7, 7, 8, 9 \rangle.$$

◯ Insertion-sort ◯ Bubble-sort $\surd$ **Merge-sort**

(c) (2')

$$\langle 4, 5, 7, 6, 8, 2, 1, 3, 7, 9 \rangle,$$
$$\langle 4, 5, 7, 6, 2, 1, 3, 7, 8, 9 \rangle,$$
$$\langle 4, 5, 6, 2, 1, 3, 7, 7, 8, 9 \rangle.$$

◯ Insertion-sort $\surd$ **Bubble-sort** ◯ Merge-sort

3. (8 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 1 point if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)	(d)

(a) (2') which of the sorts are in-place sorting algorithm?

- **A. Insertion-sort**
- **B. Bubble-sort**
- C. Merge-sort
- D. None of all

(b) (2') Which of the following situations are **true** for an array of n random numbers?

- **A. The number of inversions in this array can be found by applying a recursive algorithm adapted from merge-sort in $\Theta(n \log n)$ time.**
- **B. If it has no more than n inversions, it can be sorted in $O(n)$ time.**
- **C. If it has exactly $n(n-1)/2$ inversions, it can be sorted in $O(n)$ time.**
- D. If the array is $\langle 8, 6, 3, 7, 4 \rangle$, there are 6 inversions.

(c) (2') Which of the following statements are true?

- **A. For an array of length n, in the k-th iteration of insertion-sort, finding a correct position for a new element to be inserted at takes $\Theta(k)$ time. If we use *binary-search* instead (which takes $\Theta(\log k)$ time), it is possible to optimize the total running time to $\Theta(n \log n)$.**
- B. A sorting algorithm is stable if its worst-case time complexity is the same as its best-case time complexity.
- C. Merge-sort requires $\Theta(\log n)$ extra space when sorting an array of n elements.
- D. Given 2 sorted lists of size m and n respectively, and we want to merge them to one sorted list by mergesort. Then in the worst case, we need $m + n - 1$ comparisons.

(d) (2') Choose the situation where **insertion sort** is the most suitable among insertion sort, bubble sort and merge sort. Your choice should be the one that satisfies all the special constraints and is most efficient.

- A. Sorting an array of coordinates of points $\langle (x_1, y_1), \cdots, (x_n, y_n) \rangle$ on a 2d plane in ascending order of the x coordinate, while preserving the original order of the y coordinate for any pair of elements $(x_i, y_i), (x_j, y_j)$ with $x_i = x_j$.
- B. Sorting an array that is *almost* sorted with only $n/2$ inversions.
- **C. Sorting an array on an embedded system with quite limited memory. You may only use $\Theta(1)$ extra space, but a high time cost is acceptable.**
- D. Given a database with a large number of records. You need to sort the record with high efficiency.

4. (12 points) SmallSum

Given an array $\langle a_1, \cdots, a_n \rangle$, Let

$$f_i = \sum_{j=1}^{i-1} a_j \mathbb{I}(a_j < a_i)$$

Then, we define SmallSum:

$$\mathrm{SmallSum} = \sum_{i=1}^{n} f_i$$

For example, for array $\langle 1, 3, 4, 2, 5 \rangle$:

- for element 1: $f_1 = 0$
- for element 3: $f_2 = 1$
- for element 4: $f_3 = 1 + 3 = 4$
- for element 2: $f_4 = 1$
- for element 5: $f_5 = 1 + 3 + 4 + 2 = 10$

So the SmallSum is $0 + 1 + 4 + 1 + 10 = 16$.

Most of you can come up with the $\Theta(n^2)$ solution, but let's think it in another way.

(a) (1') For the example above, for an element a_k, how many times does it add to the sum?

A. $\sum_{i=1}^{k} \mathbb{I}(a_k < a_i)$

B. $\sum_{i=1}^{k} \mathbb{I}(a_k \geq a_i)$

C. $\sum_{i=k+1}^{n} \mathbb{I}(a_k < a_i)$

D. $\sum_{i=k+1}^{n} \mathbb{I}(a_k \geq a_i)$

(b) (8') Fill in the blanks in the code snippet below. (Hint: relate this algorithm to counting inversions.)

Consider alternating the Enhance Merge Sort algorithm to solve the problem.

The solution has 2 functions:

```
int split(int arr[], int left, int right)
{
    if (left == right)
        return 0;
    int mid = (left + right) / 2;
    int result = split(arr, left, mid) + split(arr, mid + 1, right
        );
    return result + merge(arr, left, mid, right);
}
```

The following function calculates the contribution of $\langle a_{\text{left}}, \cdots, a_{\text{mid}} \rangle$ to the sum.

```
int merge(int arr[], int left, int mid, int right)
{
    int length = right - left + 1;
    int *temp = new int[length];
    int i = 0;
    int p1 = left;
    int p2 = mid + 1;
    int result = 0;
    while (p1 <= mid && p2 <= right)
    {
        if (arr[p1] < arr[p2])
        {
            // Add arr[p1]'s contribution to final answer.
            result += arr[p1] * (right - p2 + 1);
            // move the elements of arr into temp.
            // (for the following three lines, you may not fill
               all of them.)
            temp[i++] = arr[p1++];
        }
        else
        {
            // move the elements of arr into temp.
            // (for the following three lines, you may not fill
               all of them.)
            temp[i++] = arr[p2++];
        }
    }
    while (p1 <= mid)
    {
        temp[i++] = arr[p1++];
    }
    while (p2 <= right)
    {
        temp[i++] = arr[p2++];
    }
    for (int j = 0; j < length; j++)
    {
        arr[left + j] = temp[j];
    }
    delete[] temp;
    return result;
}
```

(c) (3') What's the time complexity of our algorithm? Please answer in the form of $\Theta(\cdot)$ and prove your answer.

Solution: $\Theta(n\log(n))$

Proof: It takes $\log_2(n)$ times to split the array into single elements, which means there are $\log_2(n)$ times merging. And each merging takes $O(n)$ times to merge the array. So the time complexity is $\Theta(n\log(n))$.