Seoul National University

M1522.000900 Data Structure

Fall 2017, Kang

Homework 6: Internal Sorting (Chapter 7)

Due: November 21, 02:00 PM

Reminders

- The points of this homework add up to 100.
- Like all homeworks, this has to be done individually.
- Lead T.A.: Hyunsik Jeon (jeon185@gmail.com)
- Please type your answers in English. Illegible handwriting may get no points, at the discretion of the graders.
- If you have a question about assignments, please upload your question in eTL.
- If you want to use slipdays or consider late submission with penalties, please note that you are allowed one week to submit your assignment after the due date.

Remember that:

- Whenever you are making an assumption, please state it clearly

Question 1

Figure 1 shows a sorting algorithm implementation. Answer the following questions [20points].

```
public static int[] sort(int[] array){
    int[] aux = new int[array.length];
    int min = array[0];
    int max = array[0];
    for (int i = 1; i < array.length; i++){
        if (array[i] < min){
          min = array[i];
        }else if (array[i] > max){
          max = array[i];
        }
    }
    int[] counts = new int[max - min + 1];
    for (int i=0; i<counts.length; i++) {
        counts[i] = 0;
    }
    for (int i = 0;  i < array.length; i++){
        counts[array[i] - min]++;
    }
    counts[0]--;
    for (int i = 1; i < counts.length; i++){
        counts[i] = counts[i] + counts[i-1];
    }
    ★
    for (int i = array.length - 1; i >= 0; i--){
        aux[counts[array[i] - min]--] = array[i];
    }

    return aux;
}
```

Figure 1. A sorting algorithm implementation

(1) If the input array is [6, 3, 2, 5, 1, 0, 5, 3, 4], what is the values of the `counts` array at the star point [5points]?

(2) Let the size of the input array be *n* and the value of *(max-min)* be *k*. What is the time complexity of the algorithm [5points]?

(3) What is the space complexity of the algorithm [5points]?

(4) A sorting algorithm is said to be stable if the original ordering for duplicate keys is preserved. Is this algorithm stable [5points]?

Question 2

Figure 2 shows an optimized `MergeSort` implementation. `InsertionSort` is called when a sublist is smaller than some threshold. Assume that the number of elements in array `A` is an exponent of 2. If there are *k* calls to `MergeSort`, how many calls will there be to `InsertionSort`? Justify your answer [20points].

```
void mergesort(E[] A, E[] temp, int l, int r){
    int i, j, k, mid = (l+r)/2;
    if (l == r) return;
    //Sort the left sublist.
    if ((mid-l) >= THRESHOLD) mergesort(A, temp, l, mid);
    else inssort(A, l, mid-l+1);
    //Sort the right sublist.
    if ((r-mid) > THRESHOLD) mergesort(A, temp, mid+1, r);
    else inssort(A, mid+1, r-mid);
    //Merge
    for (int i=l; i<=mid; i++)
       temp[i] = A[i];
    for (int j=1; j<=r-mid; j++)
       temp[r-j+1] = A[j+mid];
    for (i=l,j=r,k=l; k<=r; k++){
       if (temp[i].compareTo(temp[j])<0) A[k] = temp[i++];
       else A[k] = temp[j--];
    }
}
```

Figure 2. Optimized `MergeSort` implementation

Question 3

Here is an array of 5-digit numbers:

57616 15873 22147 14562 33579 13215 12256 79514

`RadixSort` requires 5 passes for sorting 5-digit numbers. What is the result of the 3rd pass [20points]?

Question 4

`QuickSort` solves the problem by dividing the list into two sub-lists recursively. Assume that the ratio of the two sub-lists is always 1:99. What is the time complexity of sorting? Justify your answer [20points].

Question 5

`InsertionSort` can sort an array in short time when the array is nearly sorted. Therefore, taking advantage of `InsertionSort` in `QuickSort` can result in better performance. If the size of the sublist is less than k, `QuickSort` calls `InsertionSort` instead of itself. This algorithm is called `optimized-QuickSort`. Answer the following questions [20points].

Assume that time complexity of `QuickSort` and `InsertionSort` is $c_1 n \log n$, $c_2 n^2$ respectively. c_1 is the constant factor in `QuickSort`, c_2 is the constant factor in `InsertionSort`, and n is the size of the input array.

(1) What is the time complexity of `optimized-QuickSort` [10points]?

Express it using c_1, c_2, n, k.

(2) Find the condition that `optimized-QuickSort` can be faster than `QuickSort` [10points].

Express it using c_1, c_2, n, k.

(*hint*: the parameter that can be changed is only k.)