The University of Newcastle
School of Information and Physical Sciences

COMP2230/6230 Algorithms

Tutorial Week 7

30th August – 3rd September 2021

Tutorial

1. The following algorithm (Algorithm 5.1.4 from the text) is an algorithm for tiling a deficient plane with L shaped tiles (trominoes). Give a recurrence relation for the running time of the algorithm. Give the asymptotic solution for this recurrence relation.

```
Input Parameters: n, a power of 2 (the board size);
the location L of the missing tile
Output Parameters: none

tile(n,L)
{
   if  (n == 2){
       //the  board  is  a  right  tromino  T
       tile  with  T
       return
    }
    divide  the  board  into  four  n/2  by  n/2  subboards
    place  one  tromino  in  the  center
    //this  tile  is  placed  so  that  each  subboard  has  a
    //missing  square
    //let m[i] denote the location of the missing squares
    for  i  =  1  to  4  do
         tile(n/2, m[i])
}
```

2. Binary insertion sort sorts the array a[1, . . . , n] by using a binary search to determine where to place the next element to be sorted. That is, if the array is sorted up to index k, then a binary search is performed to determine where a[k + 1] should be inserted. What is wrong with the following statement? Since binary search is $\Theta(\log n)$ in the worst case, and there are $n - 1$ elements to insert, the worst case time of binary insertion sort is $\Theta(n \log n)$.

3. Counting sort

```
Input Parameters: a,m
Output Parameters: a

counting_sort(a,m)
{
    // set c[k] = the number of occurrences of value k
    // in the array a.
    // begin by initializing c to zero.
    for k = 0 to m
        c[k] = 0
    n = a.last
    for i = 1 to n
        c[a[i]] = c[a[i]] + 1
    // modify c so that c[k] = number of elements ≤ k
    for k = 1 to m
        c[k] = c[k] + c[k - 1]
```

```
        // sort a with the result in b
        for i = n downto 1 {
            b[c[a[i]]] = a[i]
            c[a[i]] = c[a[i]] - 1
        }
        // copy b back to a
        for i = 1 to n
            a[i] = b[i]
    }
```

Trace the counting sort for the following array: 15 3 17 2 9 10 8 10

4. Consider the following version of the counting sort.

```
counting_sort(a,m)
{
    for k = 0 to m - 1
        c[k] = 0
    for i = 1 to n
        c[a[i]] = c[a[i]] + 1
    for k = 1 to m
        c[k] = c[k] + c[k - 1]
    for i = 1 to n {
        b[c[a[i]]] = a[i]
        c[a[i]] = c[a[i]] - 1
     }
    for i = 1 to n
       a[i] = b[i]
}
```

Is this algorithm correct? If so, is it stable?

Homework

5. Write an algorithm that takes an array T [1, . . . , n] of real numbers and a real number v, and returns true if there exists two indices i and j such that T [i] +T [j] = v and false otherwise. What is the running time of your algorithm?

6. Write an algorithm that computes the union A ∪ B of two n-element sets of real numbers. The sets are represented as sorted arrays (there are no duplicates within a set). The result should be a sorted array with no duplicates. What is the running time of this algorithm?

7. Show how insertion sort works on the following array:

   14 40 31 28 3 15 17 51

8. Show how the partition algorithm from quicksort partitions the array in workshop question 7.

9. Trace the counting sort for the following array: 15 3 17 2 9 10 8

Extra Questions

10. Prove that the running time of the insertion sort in the worst case is Θ(n2), by proving both O and Ω.

11. Prove that the running time of the quicksort in the worst case is $\Theta(n2)$, by proving both O and Ω.

12. Show that any sorting algorithm that moves data only by swapping adjacent elements has worst cast time $\Omega(n2)$.

13. Show that if the size of the array is smaller than some number s, which is dependent on the implementation, system, etc., that insertion sort can run faster than mergesort. Modify mergesort to take advantage of this so that mergesort runs faster.

14. What is the time of quicksort when the values of all the data are equal? Show how mergesort merges the array in exercises 5.2.1-5.2.4 in the text.

15. Write a non-recursive version of mergesort.

16. Trace counting sort for the following arrays:

    a. 4 4 4 4 4 4
    b. 2 3 4 5

17. Trace the radix sort for each of the following arrays:
    a. 34 9134 20134 29134 4 134
    b. 4 34 134 9134 20134 29134

18. What is the worst-case time of radix sort if we replace counting sort by insertion sort? By mergesort?