Homework 3

Chapter 8 ~ Chapter 9

1. What is the smallest possible depth of a leaf in a decision tree for a comparison sort?

   ANSWER: **n-1 for an input array of length n.**

   In a decision tree, each internal node is annotated by i : j for some i and j in the range 1<=i,j<=n, where n is the no of elements in the input sequence.

   The absolute best case happens for a path in decision tree, in which, for each node i : j there is no other node i : k or k : j in the path for some k in {1,2,3,⋯..n} -{i,j}. The length of the path, thus, will be n-1.

   (Ans: n-1 即對 n 個 element 來說，對一個 leaf 而言，best case 為去比較 sort 好的數列，只 要比較 n-1 次就可以判斷大小順序完成 sorting，其 depth 為 n-1(假設 root depth 為 0)。)

2. Which of the following represents the height of a decision tree that sorts n elements?

   A) ***Ω (n log n)***
   B) ***Ω (log n)***
   C) ***Ω (n)***
   D) ***Ω (n)²***

   Solution:
   Ω (n log n)

3. Please complete the pseudocode for Counting-sort.

   Solution:

   COUNTING-SORT (A, B, n, k)
   for i ← 0 to k
   　　do C[i] ← 0
   for j ← 1 to n
   　　do C[A[j]] ← _____(1)_____

for i ← 1 to k

do C[i] ← C[i] + C [i − 1]

for j ← n downto 1

do ____(2)____ ← A[j]

C[A[j]] ← ____(3)____

1) C[A[j]] + 1,

2) B[C[A[j]]]

3) C[A[j]] − 1

4. Please use the **Counting Sort Algorithm** to sort the following array:

$$A = [2, 8, 1, 5, 2, 8, 2, 1, 9, 3, 1]$$

Solution:

range=9

C=

0	1	2	3	4	5	6	7	8	9
0	3	3	1	0	1	0	0	2	1

C=

0	1	2	3	4	5	6	7	8	9
0	3	6	7	7	8	8	8	10	11

A=

2	8	1	5	2	8	2	1	9	3	1

C=

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	3	6	7	7	8	8	8	10	11
	2	5	6		7			9	10
	1	4							
	0	3							

B=

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	**1**	**1**	**2**	**2**	**2**	**3**	**5**	**8**	**8**	**9**

5. Explain why it is said that **Counting Sort Algorithm** is stable.

Answer:

"In the COUNTING-SORT algorithm, when there are two elements with the same value, the element that appears later in the input array is always placed first in the sorted output array. As a result, the element that appears first in the input array is always assigned a larger position in the output array. This is because the value of (C[A[j]]) is decremented by 1 each time an element (A[j]) is placed in the output array, ensuring that the relative order of equal elements is preserved."

6. Illustrate the operation Radix-sort on the following list of number:
   $[23, 672, 89, 456, 123, 987, 784]$.

index	**Iteration 1**	**Iteration 2**	**Iteration 3**

0			**023,089**
1			**123**
2	**672**	**23,123**	
3	**23,123**		
4	**784**		**456**
5		**456**	
6	**456**		**672**
7	**987**	**672**	**784**
8		**784,987,89**	
9	**89**		**987**

7. Explain how radix sort violates the ground rules for a comparison sort?

   Answer:

   - Using counting sort allows us to gain information about keys by means other than directly comparing 2 keys.
   - Used keys as array indices.

8. Which of the following sorting algorithms are unstable: insertion sort, merge sort, heapsort, and quicksort?

   Answer: Heapsort, Quicksort

9. Please complete the pseudocode for Bucket-Sort.

   Solution:

   BUCKET-SORT (A, n)

   Let B [0..n − 1] be a new array

   n=A.length

   for i = 0 to **n – 1**

   make B[i] an empty list

   for i ← **1 to n**

   do insert A[i] into list **B [[n A[i]]]**

   for i ← 0 to n − 1

do sort list B[i] with insertion sort

concatenate lists B[0], B[1], …, B[n — 1] together in order

return the concatenated lists

1) **n – 1**
2) **1 to n**
3) **B [[n A[i]]]**

10. Illustrate the operation of BUCKET-SORT on the array.

$$A = [.56, .78, .90, .67, .45, .54, .32, .76, .23]$$

A=

0	1	2	3	4	5	6	7	8
.56	**.78**	**.90**	**.67**	**.45**	**.54**	**.32**	**.76**	**.23**

B [.0-.111]
/

B [0.111 -0.222]
/

B [0.222 -0.333]
.32,.23

B [0.333-0.444]
/

B [0.444-0.555]
.45,.54

B [0.555-0.666]
.56

B [0.666-0.777]
.67,.76

B [0.777-0.888]
.78

B [0.888-1]
.90

Sort B[i] list sort

Concatenate sorted B lists	
0 - 0.111	/
0.111 - 0.222	/
0.222 – 0.333	.23,.32
0.333-0.444	/
0.444-0.555	.45,.54
0.555-0.666	.56
0.666-0.777	.67,.76
0.777-0.888	.78
0.888-1	.90

Return the concatenated and sorted lists **A= <.23,.32,.45,.54,.56,.67,.76,.78,.90>**

11. (10pt.) Given an immutable *int* array *arr* with length k, where $k > 0$. Please design a C program (or pseudo code) **at most $3\lfloor k/2 \rfloor$ comparisons** to find both minimum and maximum. (Please finish the function *findMinAndMax*)

```c
#include<stdio.h>

void findMinAndMax(const int* arr, int k, int* max, int* min){
    // TODO
}

int main(){
    int a[] = {/* int array */};
    int max, min;
    findMinAndMax(a, sizeof(a)/sizeof(a[0]), &max, &min);
    printf("maximum: %d\n", max);
    printf("minimum: %d\n", min);
    return 0;
}
```

Ans: (reference)

```c
void findMinAndMax(const int* arr, int k, int* max, int* min){
    *max = arr[k-1];
    *min = arr[k-1];
    for(int i = 1; i < k; i+=2){
        if(arr[i] < arr[i-1]){
            if(arr[i-1] > *max) *max = arr[i-1];
            if(arr[i] < *min)   *min = arr[i];
        }else{
            if(arr[i] > *max)   *max = arr[i];
            if(arr[i-1] < *min) *min = arr[i-1];
        }
    }
}
```

12. (2pt.) Following the previous question. According to your program, how many times does a comparison occur if $a = [4, 1, 2, 3]$?
    6 times, some programs may require only 4 or 5 times.
13. (2pt.) Following the previous question. According to your program, how many times does a comparison occur if $a = [4, 1, 2, 3, 5]$?
    6 times.
14. (2pt.) The problem of computing the i-th smallest element (Start from 0) of an input unsorted distinct array can be solved easily in $O(n \log n)$ time using sorting, but there's potential for improvement. RANDOMIZED-SELECT is an extension of the partitioning step in quicksort and aims to efficiently determine the desired element without fully sorting the entire array. Can you finish the code below in C (or pseudo code) to implement the RANDOMIZED-SELECT algorithm? **(Please just finish Part A)**

    The function *randomizedPartition* splits the mutable integer array arr of length len at random index p into two parts, ensuring that indices less than p have values less than $arr[p]$, and indices greater than p have values not less than $arr[p]$.

```c
int randomizedSelection(int* arr, int len, int i){
    int p = randomizedPartition(arr, len);
    if(p == i){
        return /* (A) */;
    }else if(p < i){
        return randomizedSelection(/* (B) */);
    }
    return randomizedSelection(/* (C) */);
}

int main(){
    srand(time(NULL));
    int a[] = {7,1,2,5,3,6,8,10};
    int len = sizeof(a)/sizeof(a[0]);
    printf("the smallest in a is: %d\n", randomizedSelection(a, len, 0));
    printf("the 1st smallest in a is: %d\n", randomizedSelection(a, len, 1));
    printf("the 2nd smallest in a is: %d\n", randomizedSelection(a, len, 2));
    return 0;
}
```

Output:

```
the smallest in a is: 1
the 1st smallest in a is: 2
the 2nd smallest in a is: 3
```

15. (5pt.) Following the previous question. **Please finish part B.**

    $(arr + p,\ len - p,\ i - p)$ or $(arr + p + 1,\ len - p - 1,\ i - p - 1)$

16. (5pt.) Following the previous question. **Please finish part C.**

    Ans:

```c
int randomizedSelection(int* arr, int len, int i){
    int p = randomizedPartition(arr, len);
    if(p == i){
        return arr[p];
    }else if(p < i){
        return randomizedSelection(arr+p+1, len-p-1, i-p-1);
    }
    return randomizedSelection(arr, p, i);
}
```

17. (10pt.) Following the previous question. **Please finish the function *randomizedPartition*** in C (or pseudo code). (We have implemented the function *swap*, you can call it directly.)

```
void swap(int* arr, int i , int j){
    if(i != j){
        arr[i] = arr[i] ^ arr[j];
        arr[j] = arr[i] ^ arr[j];
        arr[i] = arr[i] ^ arr[j];
    }
}

int randomizedPartition(int* arr, int len){
    int pivot = rand() % len;
    swap(arr, pivot, len-1);
    // TODO
}
```

Ans:

```
int randomizedPartition(int* arr, int len){
    int pivot = rand() % len;
    swap(arr, pivot, len-1);
    int j = 0;
    for(int i = 0; i < len-1; i++){
        if(arr[i] < arr[len-1]){
            swap(arr, i, j);
            j++;
        }
    }
    swap(arr, len-1, j);
    return j;
}
```

18. (5pt.) Explanation of when the RANDOMIZED-SELECT algorithm encounters its worst-case scenario and what the worst-case running time of this algorithm is. $\Theta(n^2)$, because we could be extremely unlucky and always recurse on a subarray that is only 1 element smaller than the previous subarray.
19. (3pt.) The expected running time of RANDOMIZED-SELECT is a random variable $T(n)$, where n is the length of given array. For $k = 1,2, \dots, n$, k represents the left partitioned subarray has k elements. Define indicator random variable $X_k = I$ {the left partitioned subarray has k elements}. Indicator random variable is a special kind of random variable based on such an event, and it equals 1 if the event happens and 0 otherwise. What is the expectation of X_k, i.e. $E[X_k]$?

$$E[X_k] = \frac{1}{n}$$

20. (6pt.) The expected running time of RANDOMIZED-SELECT can be represented in a recurrence form as $T(n) \leq \Sigma_{k=1}^{n} X_k \cdot (T(\max(k-1, n-k)) + O(n))$. Please explain this recurrence form.

    Ans:

    The position of each partition may appear between $[1, n]$, X_k represents that when the partition appears at k, X_k will be 1, and it will be 0 in all other cases. Therefore, when using Σ to calculate the sum, we can precisely consider different partition scenarios. Since each partition requires traversing the entire array, the time complexity is $O(n)$. Due to the uncertainty of whether to choose the left $k-1$ subarray or the right $n-k$ subarray for the next iteration, we choose the maximum one.

Reference code for 11-17

https://gist.github.com/ksw2000/1a950e717bfdfa477d3018bb75f17df1