Ch. 7.7-10 Sorting 習題

Q1:

- What is time complexity of LSD (Least Significant Digit) radix sort?

Ans 1:

- O(d*(n+r))

- r: number of buckets
- n: Amount of data
- d: digits base on the base.
  - i.e. b = 10. k is the maximum possible value, then d would be $O(\log_{10}(k))$.
- Subroutine: Counting sort or Bucket sort
- d rounds base on digits.

Q2:

- What is space complexity of LSD radix sort?

Ans 2:

- $O(n + r)$

- r: number of buckets
- n: link pointers

Q3:

- Is Radix Sort a stable sort? Why?

Ans 3:

- Radix Sort is stable.
- The radix sort algorithm sorts number by 1 digit at a time. For each d round, the number appearing before would never move to the number appearing later. This ensures that the number appearing before other numbers in the input array would maintain that same order in the final.

Q4:

- Give an application example for the linear-time sorting algorithm.

Ans 4:

- Radix sort application:
  - English dictionary base on alphabet radix
  - Library books index.

Q5:

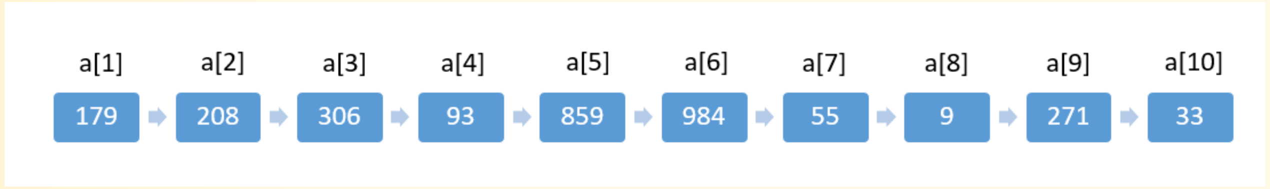
- (1) Define what does “Sorting on several keys” mean.
- (2) For the sorting algorithm on multiple keys, please propose an application example other than the sorting deck of cards in the textbook.

Ans 5:

1) Sorting on several keys: A list of records is said to be sorted with respect to the keys $K1$, $K2$, ... , Kr iff for every pair of records i and j, $i < j$ and $(K1,K2,...,Kr) \leq (K1,K2,...,Kr)$. We say $(x_1,...,x_r) \leq (y_1,...,y_r)$ iff either $x_k = y_k$, $1 \leq k \leq n$, and $x_{n+1} < y_{n+1}$ for some $n < r$, or $x_k = y_k, 1 \leq k \leq r$
2) English dictionary.

Q6:

1) We provide the LSD Radix Sort code(in next page, originated form the textbook) below. Please explain what does the code of RadixSort do. You may use the following initial values of the a[] array elements to explain the algorithm.



a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
179	208	306	93	859	984	55	9	271	33

2) Write the status of the list (12, 2, 16, 30, 8, 28, 4, 10, 20, 6, 18) at the end of each pass of RadixSort. Use r=10.

```
template<class T>
int RadixSort(T *a, int *link, const int d, const int r, const int n){
    // Sort a[1:n] using a d-digit  radix r sort. digit(a[i],j,r) returns the jth radix-r
    // digit(from the left) of a[i]'s key. Each digit is in the range is [0,r).
    // Sorting within a digit is done using a bin sort
    int e[r],f[r];
    int first=1;
    for(int i=0;i<n;i++) link[i]=i+1;
    link[n]=0;

    for(i=d-1;i>=0;i--){
        // Sort on digit i
        fill(f,f+r,0); // initialize bins to empty queues
        for(int current=first;current;current=link[current]){
            //put records into queue/bin
            int k=digit(a[current],i,r);
            if(f[k]==0) f[k]=current;
            else link[e[k]]=current;
            e[k]=current;
        }
        for(j=0;!f[j];j++); // Find first nonempty queue/bin
        first=f[j];
        int last=e[j];
        for(int k=j+1;k<r;k++){
            if(f[k]){
                link[last]=f[k];
                last=e[k];
            }
        }
        link[last]=0;
    }
    return first;
}
```

Implementation detail: https://www.geeksforgeeks.org/radix-sort/

Ans 6 (1) :

- That link is an array that indicate the next address of next value. The “first” is the first index from the unsorted array.

- Pass 1: 271 93 33 984 55 306 208 179 859 9
- Pass 2: 306 208 9 33 55 859 271 179 984 93
- Pass 3: 9 33 55 93 179 208 271 306 859 948

Ans 6 (2):

- 12, 2, 16, 30, 8, 28, 4, 10, 20, 6, 18

- Step 1: 30, 10, 20, 12, 2, 4, 16, 6, 8, 28, 18
- Step 2: 2, 4, 6, 8, 10, 12, 16, 18, 20, 28, 30

Q7:

- Please propose a hybrid sorting algorithm and discuss how and why the algorithm works, particularly explain when and why one algorithm is switched to another one in the hybrid approach. You may go online and search for Hybrid Quick Sort Algorithm (Quick Sort + Insertion Sort) to learn how people discuss the working principle of the algorithm.

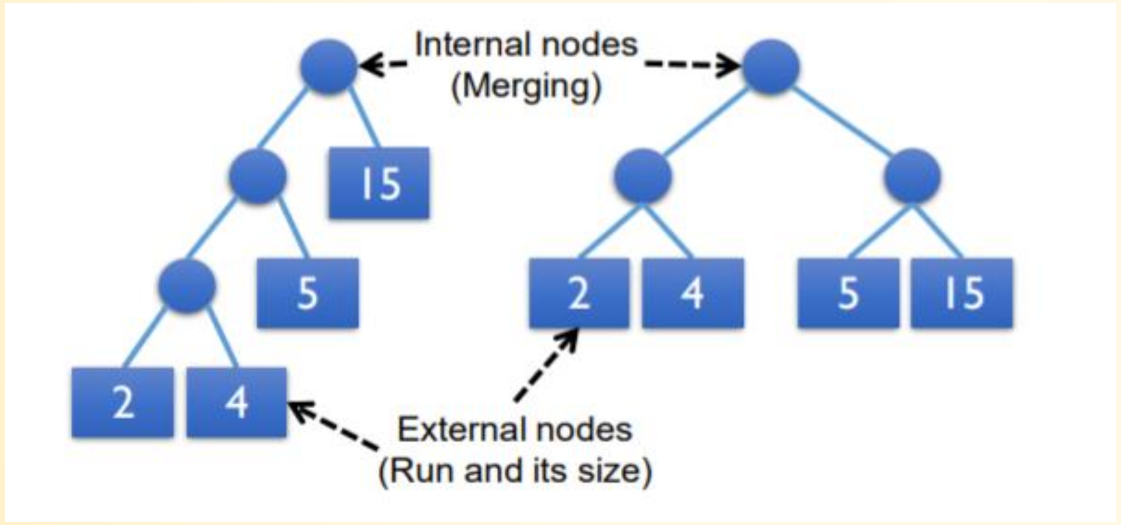
Ans 7:

- Quick Sort: one of fastest sorting algorithm for large inputs in average case because of tail recursive.

- When dividing elements less than threshold, switch the sorting methods to insertion sort because it has good performance at small inputs.

Q8:

- (1) What does “Optimize Merge” mean?
- (2) Why should we categorize nodes into external nodes and internal nodes?



Ans 8 (1):

- Optimize merge: make the cost of merge tree minimized.

- Tree left $Cost$ = (2 + 4) + (2 + 4 + 5) + (2 +4+5+15) =$2*3$+$4*3$+$5*2$ +$15*1$ = 43

- Tree right $Cost$ = $2*2$+$4*2$+$5*2$ + $15*2$ = 52

Ans 8 (2):

- external nodes: runs before merge.
- Internal nodes: merge results from several runs.

Q9:

- Please complete the following table and try to give examples to the worst and best time complexity cases.

Algorithm	Time Complexity		
	Best	Average	Worst
Insertion Sort			
Quick Sort			
Heap Sort			

Ans 9:

Algorithm	Time Complexity		
	Best	Average	Worst
Insertion Sort	O(n)	O(n^2)	O(n^2)
Quick Sort	O(n*log(n))	O(n*log(n))	O(n^2)
Heap Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))

Q10:

- Please give an example of the external sorting.
- (1) Introduce the concept and practical application of the algorithm.
- (2) Give a brief introduction to the implementation of the algorithm

Ans 10:

- (1) External sorting is sorting algorithms that can handle massive amounts of data. External sorting is worked when the data don’t fit into the main memory
- (2) External merge sort algorithm, which sorts chunks that each fit in RAM, then merges the sorted chunks together.