HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

C Programming Basic

Sorting - part 2

CONTENT

- Data generation
- Implement merge sort, quick sort, heap sort algorithms
- Experiments

Exercise 1: data generation

- Profile of each staff has following information
  - Full name
  - Date of birth
- Write a program that generates *n* profiles randomly and store to a file profile-n.txt under the format:
  - Line 2*i*-1 and 2*i* (*i* = 1,…, *n*) respectively write the full name and date of birth of the profile *i*. Full name has the format <last_name> <middle_name> <first_name> and the date of birth has the format YYYY-MM-DD
  - Line 2*n*+1: write the character # to indicate the end of the data

Profile-5.txt

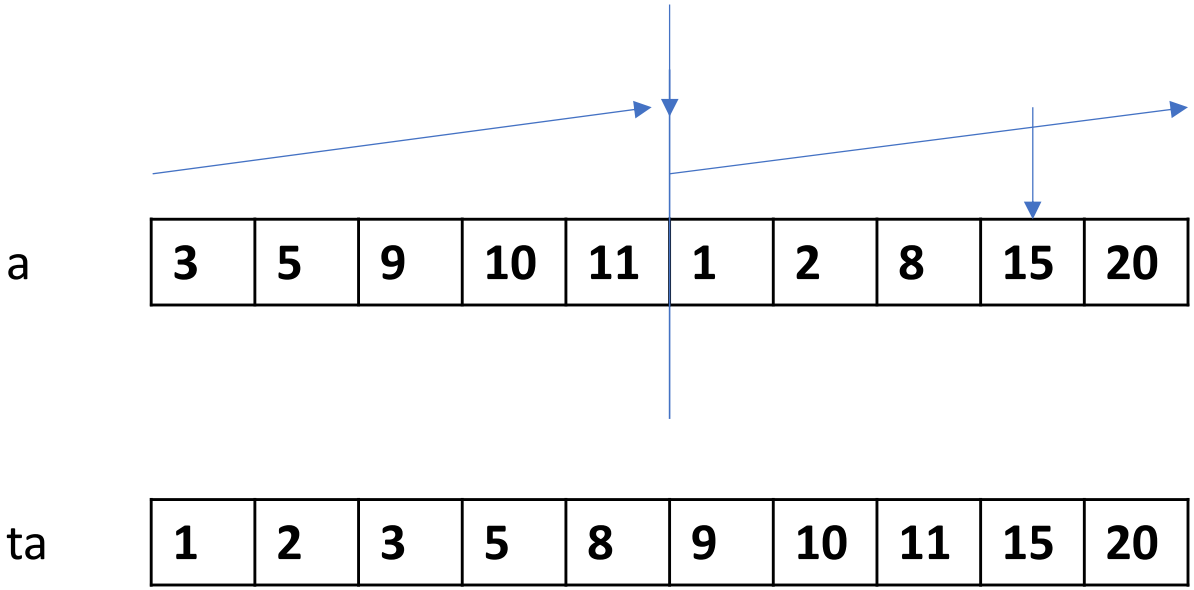
```
Bui Hai An
1980-02-30
Pham Viet Anh
1986-10-08
Do Duc Bang
1990-04-24
Dang Van Cuong
1987-08-17
Pham Viet Anh
1986-05-20
#
```

Merge sort

- Problem: Sorting *n* elements of an array S[0]…S[*n*-1]
- Merge-sort on an input sequence ***S*** with ***n*** elements consists of three steps:
  - Divide: partition ***S*** into two sequences S_1 and S_2 of about ***n***/2 elements each
  - Conquer: recursively sort S_1 and $S_2$by using merge sort
  - Combine: merge S_1 and S_2 into a unique sorted sequence

```
void mergeSort(int A[], int L, int R) {
  if(L < R){
    int M = (L+R)/2;
    mergeSort(A,L,M);
    mergeSort(A,M+1,R);
    merge(A,L,M,R);
  }
}
```

Merge sort



Merge sort

- Use auxiliary array
- Running time
  - Worst case O(*n*log*n*)
  - Best case: O(*n*log*n*)

```
void merge(int A[], int L, int M, int R) {
// tron 2 day da sap A[L..M] va A[M+1..R]
  int i = L; int j = M+1;
  for(int k = L; k <= R; k++){
    if(i > M){ TA[k] = A[j]; j++;}
    else if(j > R){TA[k] = A[i]; i++;}
    else{
      if(A[i] < A[j]){
         TA[k] = A[i]; i++;
      }
      else {
         TA[k] = A[j]; j++;
      }
    }
  }
  for(int k = L; k <= R; k++) A[k] = TA[k];
}
```

Exercise 2

- Write a program that reads a sequence of n profiles from data files generated from the exercise 1 above, sort the sequence in non-decreasing order (prioritize full name, then date of birth) by the merge sort algorithm
- Input (profile-n.txt)
  - Line 2*i*-1 and 2*i* (*i* = 1,…, *n*) respectively write the fullname and date of birth of the profile *i*. Fullname has the format <last_name> <middle_name> <first_name> and the date of birth has the format YYYY-MM-DD
  - Last line: write the character # to indicate the end of the data
- Result (sorted-profile-n.txt)
  - Write the sorted sequence under the format:
    - Line 2*i*-1 and 2*i* (*i* = 1,…, *n*) respectively write the fullname and date of birth of the profile *i*. Fullname has the format <last_name> <middle_name> <first_name> and the date of birth has the format YYYY-MM-DD
    - Last line: write the character # to indicate the end of the data

Quick sort

The quick sort algorithm is described recursively as following (similar to merge sort):

1. **Base case**. If the array has only one element, then the array is sorted already, return it without doing anything.
2. **Divide**:
   - Select an element in the array, and call it as the pivot ***p***.
   - Divide the array into 2 subarrays: Left subarray (*L*) consists of elements ≤ the pivot, right subarray (*R*) consists of elements ≥ the pivot. This operation is called "Partition".
3. **Conquer**: recursively call QuickSort for 2 subarrays $L = A[p \ldots q]$ and $R = A[q+1 \ldots r]$.
4. **Combine**: The sorted array is *L p R*.

In contrast to Merge Sort, in Quick Sort: division operation is complicate, but the Partition operation is simple.

Quick sort

```
void quickSort(int A[], int L, int R) {
  if(L < R){
    int index = (L + R)/2;
    index = partition(A, L, R, index);
    if(L < index)
      quickSort(A, L, index-1);
    if(index < R)
      quickSort(A, index+1, R);
  }
}
```

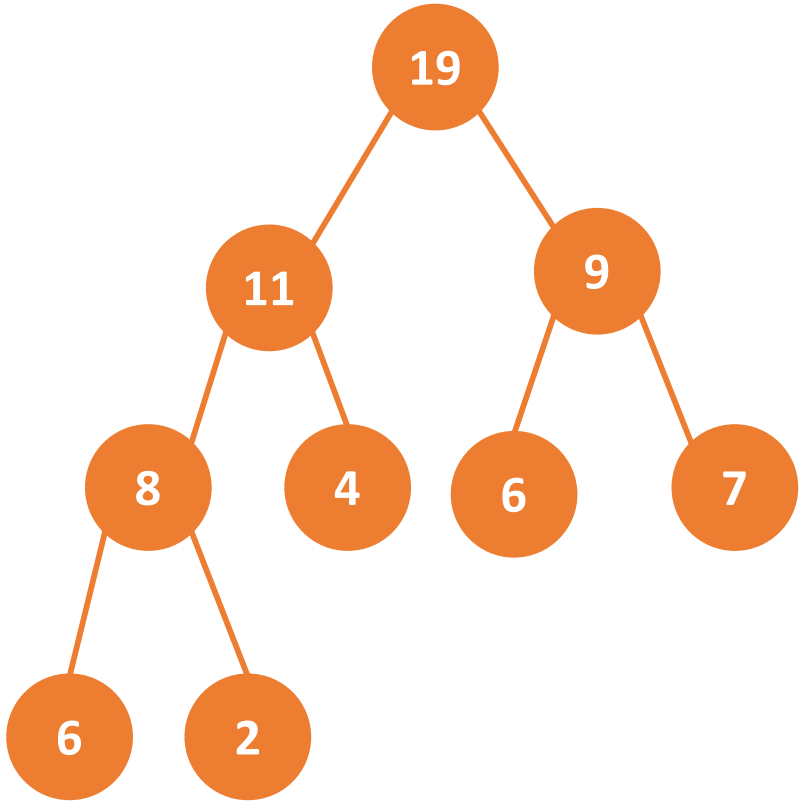
```
int partition(int A[], int L, int R, int
                          indexPivot) {
  int pivot = A[indexPivot];
  swap(A[indexPivot], A[R]);
  int storeIndex = L;
  for(int i = L; i <= R-1; i++){
    if(A[i] < pivot){
      swap(A[storeIndex], A[i]);
      storeIndex++;
    }
  }
  swap(A[storeIndex], A[R]);
  return storeIndex;
}
```

Exercise 3

- Write a program that reads a sequence of n profiles from data files generated from the exercise 1 above, sort the sequence in non-decreasing order (prioritize full name, then date of birth) by the quick sort algorithm
- Input (profile-n.txt)
  - Line 2*i*-1 and 2*i* (*i* = 1,…, *n*) respectively write the fullname and date of birth of the profile *i*. Fullname has the format <last_name> <middle_name> <first_name> and the date of birth has the format YYYY-MM-DD
  - Last line: write the character # to indicate the end of the data
- Result (sorted-profile-n.txt)
  - Write the sorted sequence under the format:
    - Line 2*i*-1 and 2*i* (*i* = 1,…, *n*) respectively write the fullname and date of birth of the profile *i*. Fullname has the format <last_name> <middle_name> <first_name> and the date of birth has the format YYYY-MM-DD
    - Last line: write the character # to indicate the end of the data

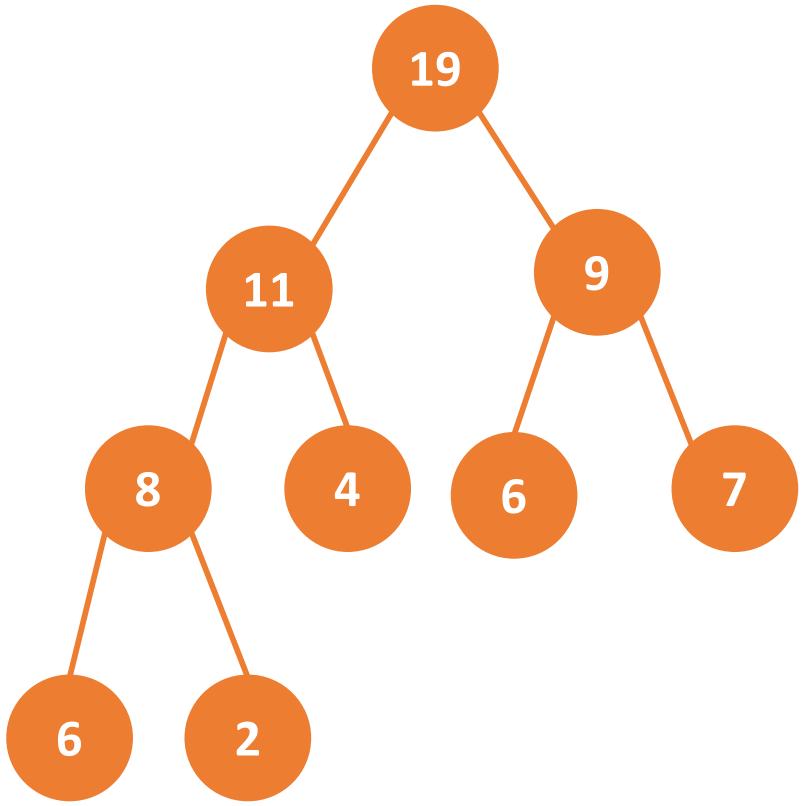
Heap sort

- Heap structure (max-heap)
  - Complete tree
  - Key of each node is greater or equal to the keys of its children (max-heap property)
- Map the sequence A[1…N] to a complete tree
  - Root is A[1]
  - A[2i] and A[2i+1] are respectively the left child and the right child of A[i]
  - The height of the tree is log$N$ + 1



Heap sort

- Heapify
  - Status:
    - max-heap property at A[i] is destroyed
    - max-heap property at children of A[i] is satisfied
  - Adjust the tree to recover the max-heap property at the root A[i]



Heap sort

- Running time: O(log*N*)

```
void heapify(int A[], int i, int N) {
  int L = 2*i;
  int R = 2*i+1;
  int max = i;
  if(L <= N && A[L] > A[i])
    max = L;
  if(R <= N && A[R] > A[max])
    max = R;
  if(max != i){
    swap(A[i], A[max]);
    heapify(A,max,N);
  }
}
```

Heap sort

- Heap sort
  - Build max-heap
  - Swap A[1] and A[N]
  - Heapify at A[1] until A[1..N-1]
  - Swap A[1] and A[N-1]
  - Heapify at A[1] until A[1..N-2]
  - …
- Run time: O(NlogN)

```
void buildHeap(int A[], int N) {
  for(int i = N/2; i >= 1; i--)
    heapify(A,i,N);
}
void heapSort(int A[], int N) {
  // index tu 1 -> N
  buildHeap(A,N);
  for(int i = N; i > 1; i--) {
    swap(A[1], A[i]);
    heapify(A, 1, i-1);
  }
}
```

Exercise 4

- Write a program that reads a sequence of n profiles from data files generated from the exercise 1 above, sort the sequence in non-decreasing order (prioritize full name, then date of birth) by the heap sort algorithm
- Input (profile-n.txt)
  - Line 2*i*-1 and 2*i* (*i* = 1,…, *n*) respectively write the fullname and date of birth of the profile *i*. Fullname has the format <last_name> <middle_name> <first_name> and the date of birth has the format YYYY-MM-DD
  - Last line: write the character # to indicate the end of the data
- Result (sorted-profile-n.txt)
  - Write the sorted sequence under the format:
    - Line 2*i*-1 and 2*i* (*i* = 1,…, *n*) respectively write the fullname and date of birth of the profile *i*. Fullname has the format <last_name> <middle_name> <first_name> and the date of birth has the format YYYY-MM-DD
    - Last line: write the character # to indicate the end of the data