Assignment of

Algorithms and Advanced C Programming

Torino 2015
Y. Duan

Part I

Advanced C Programming

Chapter 1

C Basics

Text

Ex.1 Declare an array of integers and search a number in the array. Write the C program.

Ex.2 Try to improve the efficiency of the searching (by decreasing the times for searching). Write the C program.

Ex.3 Generate the first 20 numbers of the Fibonacci sequence which starts with 1 and 2.

Ex.4 Let the user enter 10 numbers and save their absolute values in an array. Find the maximum number (only 1) in the array, its position and the average of all the numbers.

Ex.5 Based on the previous problem, also find the number with the maximum occurrences. Example

6	4	7	5	1	6	3	4	6	5

Max occurrences: 6 Times: 3

Ex.6 Write a C program to implement Insertion Sort.

Ex.7 Write a function that calculates the factorial of a number n. The function receives the number n as the argument and returns the factorial.

Ex.8 Write a C program where the user inputs 10 numbers and calculate how many *prime* numbers there are. Testing if a number is prime should be written in a function like `int isPrime(int n)` which returns 1 if n is prime, otherwise 0.

Ex.9 Write a function that reverse the numbers in an array. For example, $[2, 0, 1, 5]$ is reversed to $[5, 1, 0, 2]$. Do not use additional space.

Ex.10 Write a function that takes an array as parameter, calculates the sum of the absolute values of all the numbers in the array and returns the sum.

Ex.11 Write a function that finds the maximum value in an array.

Ex.12 Let the user enter N types fruits with their prices. Find the fruit with the highest price and print the result. For example, the user enters: The

apple	5
banana	6
cherry	3
elderberry	5
date	4

program should print: `The most expensive fruit is banana and the price is 6.`

Ex.13 Write a function like `int findchar(char *s, char c)` that finds how many character `c` there are in the string `s`.

Ex.14 Write a function like `int contains(char *a, char *b)` that returns 1 if `b` is a sub-string of `a`, otherwise 0. E.g. `"hongrizhihuo"` contains `"rizhi"`.

Ex.15 Write a function that reverses a string. E.g. `italy` will be reversed to `ylati`.

Ex.16 Write a C program to compute the product of two matrices and print the result.

Ex.17 Assume we have a map of size $M \times M$. Let the user draw 5 rectangles on the map by inserting the coordinates of the upper left corner and the lower right corner. Find if there are overlapping rectangles. If yes, find the number of layers at the thickest part.

Ex.18 Find the peaks in an 2-d array. A peak is an element where its value is larger than all the values of the elements around it. For example, the following array has 2 peaks, which are a[1][1] and a[2][3].

$$\begin{matrix} 8 & 2 & 3 & 4 \\ 5 & 9 & 7 & 2 \\ 0 & 1 & 1 & 8 \end{matrix}$$

Ex.19 A file stores the exam results of the students in a class in the following format:

There are at most 40 students in the class. Write a C programm to:

(a) calculate the average score of the c;

(b) find the student with the maximum score and print out his/her name, nationality and the score;

Harry	Britain	100
Jim	Korea	90
Peter	Britain	110
Bob	USA	90
...	...	...

(c) find the country with the highest average score.

Ex.20 In the previous problem, sort the students according to their marks and write the results to another file whose name is passed from the command line.

Chapter 2

Recursion

Ex.1 Calculate recursively the factorial of a number n.

Ex.2 Recursively print the following graph which is composed of stars.

```
* * * * *
* * * *
* * *
* *
*
```

Ex.3 Recursively print the following graph which is composed of stars.

```
*
* *
* * *
* * * *
* * * * *
```

Ex.4 Print a string reversely in a recursive way. Example: given a string `"yongyuanpeibanni"` the program should print `innabiepnauygnoy`.

Chapter 3

Struct

Ex.1 Define a `struct` to describe a car that has the *price*, *brand status* (new or used). Let the user enter 5 cars. Sort the cars by their prices. Then write a function that finds the cheapest new car.

Chapter 4

Dynamic Allocation

Ex.1 Implement the counting sort. Let the user enter the numbers. If needed, always use dynamic allocation.

Ex.2 In **Ex.1** of Chapter 3, use dynamic allocation to reserve the memory for 5 cars, instead of declaring an array.

Ex.3 Write a function as
`void addition (int *a, int *b, int len, int **sum, int **diff)`
that finds the sum and difference of two integers stored in two arrays `a` and `b`. The function should dynamically allocate two arrays of length `len+1` and store the result in `sum` and `diff`. Assume the integer stored in `a` is always larger than the one in `b`.

$$\begin{array}{cccccc} & 4 & 9 & 9 & 1 & \text{(a)} \\ + & 0 & 2 & 3 & 0 & \text{(b)} \\ \hline 0 & 5 & 2 & 2 & 1 & \\ \multicolumn{6}{c}{\text{(sum)}} \end{array} \qquad \begin{array}{cccccc} & 4 & 9 & 9 & 1 & \text{(a)} \\ - & 0 & 2 & 3 & 0 & \text{(b)} \\ \hline & 4 & 7 & 6 & 1 & \\ \multicolumn{6}{c}{\text{(diff)}} \end{array}$$

Chapter 5

List

Ex.1 Let the user keep entering numbers until a negative number is entered. Store the numbers using list. Write a single function that finds the maximum and minimum number using both *iteration* and *recursion*.

Ex.2 Let the user keep entering cars until a car with negative price is entered. Write a function that finds the cheapest new car.

Ex.3 Write a function that finds the cheapest new car in the previous problem using *recursion*.

Ex.4 Suppose there are 10 people whose IDs are from 1 to 10. Then kill the people with the even ID. Print the alive people in the reverse order.

Chapter 6

Trees

Ex.1 Write a function `void visitLevel(node_t *root, int level)` that prints the keys of all the node at level `level`.

Ex.2 Write a function `int findKey(node_t *root, int v)` that searches a key ‘`v`’ in a tree. This function returns 1 if the searched key exists, otherwise 0.

Part II

Algorithms

Chapter 7

Complexity

Text

1. Evaluate the complexity of the following algorithm in terms of number of comparisons (C) and number of transfers(M):

```
duplicate = 0;
for (i = 0; i < n; ++i ) {
    for (j = 0; j < n; ++j ) {
        if ( i != j && A[ i ] == A[ j ] ) {
            duplicate = 1;
            break;
        }
    }
    if (duplicate == 1) {
        break;
    }
}
```

2. Try to reduce the comparison times in the code above. Evaluate the complexity of your new code and compare it with the old one.

Solution

1. Worst case analysis:

```
duplicate = 0;                                          //
for (i = 0; i < n; ++i ) {                              // n
    for (j = 0; j < n; ++j ) {                          // n * n
        if ( i != j && A[ i ] == A[ j ] ) {             // n * n * 1
            duplicate = 1;
            break;
        }
    }
    if (duplicate == 1) {
        break;
    }
}
```

$$T_{Compare}(n) = O(n^2)$$
$$T_{Assignment}(n) = O(n^2)$$

2. Improvement

```
    duplicate = 0;                          //
    for (i = 0; i < n; ++i ) {              // T1(n) = n
        // T2(n) = T1(n)*[(n-1)+(n-2)+(n-3)+...+1]
        for (j = i+1; j < n; ++j ) {
            if ( A[ i ] == A[ j ] ) {       // T3(n) = T2(n) * 1
                duplicate = 1;
                break;
            }
        }
        if (duplicate == 1) {
            break;
        }
    }
```

$$T_{Compare}(n) = O(n^2)$$
$$T_{Assignment}(n) = O(n^2)$$

Chapter 8

Union Find

Text

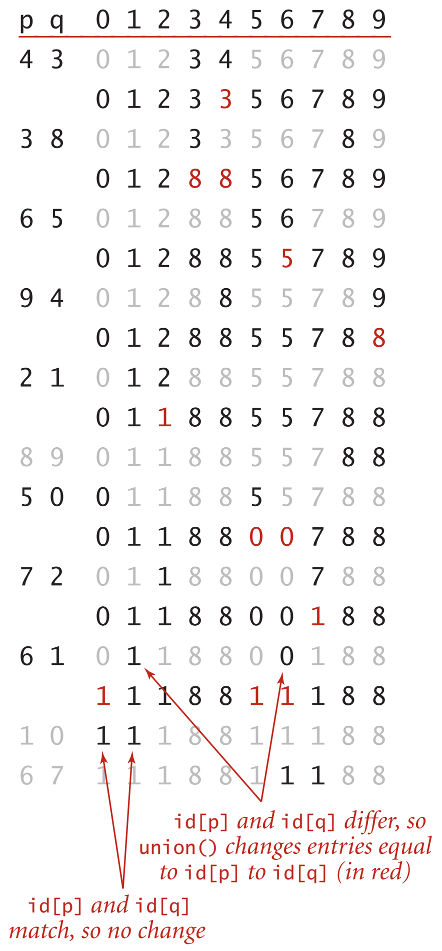
1. Given the following sequence of pairs, where the relation i-j means that node i is adjacent to node j:

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

   apply an on-line connectivity algorithm with **quick find**, **quick union** and **weighted quick union** showing at each step the contents of the array and the forest of trees at the final step. Nodes are named with integers in the range from 0 to 9.

Solution

Quick Find

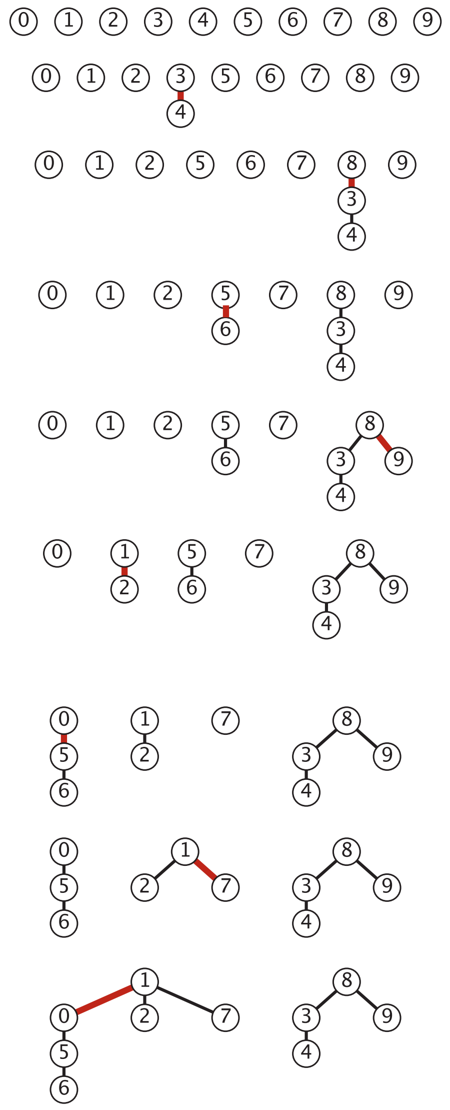


p	q	0	1	2	3	4	5	6	7	8	9
4	3	0	1	2	3	4	5	6	7	8	9
		0	1	2	3	3	5	6	7	8	9
3	8	0	1	2	3	3	5	6	7	8	9
		0	1	2	8	8	5	6	7	8	9
6	5	0	1	2	8	8	5	6	7	8	9
		0	1	2	8	8	5	5	7	8	9
9	4	0	1	2	8	8	5	5	7	8	9
		0	1	2	8	8	5	5	7	8	8
2	1	0	1	2	8	8	5	5	7	8	8
		0	1	1	8	8	5	5	7	8	8
8	9	0	1	1	8	8	5	5	7	8	8
5	0	0	1	1	8	8	5	5	7	8	8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

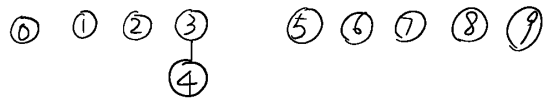
`id[p]` *and* `id[q]` *differ, so* `union()` *changes entries equal to* `id[p]` *to* `id[q]` *(in red)*
`id[p]` *and* `id[q]` *match, so no change*

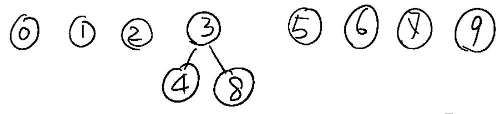
Quick Union

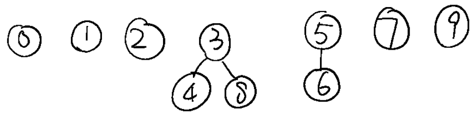
p	q	0	1	2	3	4	5	6	7	8	9
4	3	0	1	2	3	4	5	6	7	8	9
		0	1	2	3	3	5	6	7	8	9
3	8	0	1	2	3	3	5	6	7	8	9
		0	1	2	8	3	5	6	7	8	9
6	5	0	1	2	8	3	5	6	7	8	9
		0	1	2	8	3	5	5	7	8	9
9	4	0	1	2	8	3	5	5	7	8	9
		0	1	2	8	3	5	5	7	8	8
2	1	0	1	2	8	3	5	5	7	8	8
		0	1	1	8	3	5	5	7	8	8
8	9	0	1	1	8	3	5	5	7	8	8
5	0	0	1	1	8	3	5	5	7	8	8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

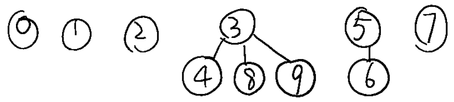


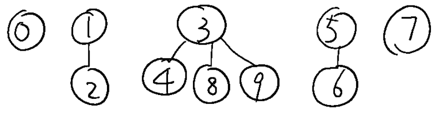
Weighted Quick Union

4-3


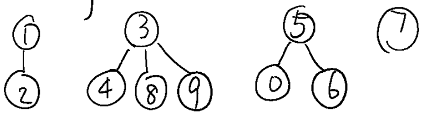
3-8


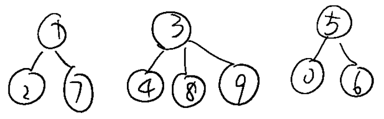
6-5


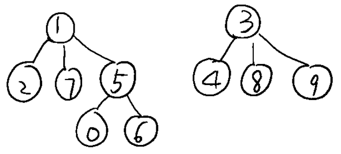
9-4


2-1


8-9 Already Connected

5-0


7-2


6-1


1-0 already Connected

6-7 already Connected

Chapter 9

Sorting Algorithms

Text

All the solutions require intermediate procedures and results.

1. Sorting the following numbers in both ascending and descending order using insertion sort:

   12 43 52 34 12 14 18 29 28 29 26 48 32 20 10

2. Write a function implementing Bubble sort and evaluate the complexity. Is it in-place and stable?

3. Sorting the following numbers in both ascending and descending order using bubble sort:

   67 8 41 70 47 48 61 30

4. Sorting the following numbers in both ascending and descending order using selection sort:

   72 73 86 55 52 13 6 15 32 53

5. Sorting the following numbers in both ascending and descending order using selection sort:

   9 54 39 44 9 54 39 44 9 54

   and show that it is not stable.

6. Write a function implementing selection sorting in C language and evaluate the complexity. Is it in-place?

7. Sorting the following numbers in ascending order using shell sort. The sequence of h is $h_{i-1} = 3 * h_i + 1$:

   76, 25, 62, 43, 96, 85, 42, 83, 16, 45, 22, 23, 36, 5, 2, 63, 56, 65, 82, 3

Solution

1. Insertion sort ascending

12	43	52	34	12	14	18	29	28	29	26	48	32	20	10
12	**43**	52	34	12	14	18	29	28	29	26	48	32	20	10
12	43	**52**	34	12	14	18	29	28	29	26	48	32	20	10
12	**34**	43	52	12	14	18	29	28	29	26	48	32	20	10
12	**12**	34	43	52	14	18	29	28	29	26	48	32	20	10
12	12	**14**	34	43	52	18	29	28	29	26	48	32	20	10
12	12	14	**18**	34	43	52	29	28	29	26	48	32	20	10
12	12	14	18	**29**	34	43	52	28	29	26	48	32	20	10
12	12	14	18	**28**	29	34	43	52	29	26	48	32	20	10
12	12	14	18	28	29	**29**	34	43	52	26	48	32	20	10
12	12	14	18	**26**	28	29	29	34	43	52	48	32	20	10
12	12	14	18	26	28	29	29	34	43	**48**	52	32	20	10
12	12	14	18	26	28	29	29	**32**	34	43	48	52	20	10
12	12	14	18	**20**	26	28	29	29	32	34	43	48	52	10
10	12	12	14	18	20	26	28	29	29	32	34	43	48	52

2. See text book.

3. Bubble sort ascending:

67	8	41	70	47	48	61	30
8	41	67	47	48	61	30	70
8	41	47	48	61	30	67	70
8	41	47	48	30	61	67	70
8	41	47	30	48	61	67	70
8	41	30	47	48	61	67	70
8	30	41	47	48	61	67	70
8	30	41	47	48	61	67	70

4. Selection sort ascending:

72	73	86	55	52	13	6	15	32	53
6	73	86	55	52	13	72	15	32	53
6	13	86	55	52	73	72	15	32	53
6	13	15	55	52	73	72	86	32	53
6	13	15	32	52	73	72	86	55	53
6	13	15	32	52	73	72	86	55	53
6	13	15	32	52	53	72	86	55	73
6	13	15	32	52	53	55	86	72	73
6	13	15	32	52	53	55	72	86	73
6	13	15	32	52	53	55	72	73	86
6	13	15	32	52	53	55	72	73	86
6	13	15	32	52	53	55	72	73	86

5. Selection sort ascending:

9	54	39	44	9	54	39	44	9	54
9	54	39	44	9	54	39	44	9	54
9	9	39	44	54	54	39	44	9	54
9	9	9	44	54	54	39	44	39	54
9	9	9	39	54	54	44	44	39	54
9	9	9	39	39	54	44	44	54	54
9	9	9	39	39	44	54	44	54	54
9	9	9	39	39	44	44	54	54	54
9	9	9	39	39	44	44	54	54	54
9	9	9	39	39	44	44	54	54	54
9	9	9	39	39	44	44	54	54	54
9	9	9	39	39	44	44	54	54	54

6. See text book.

7. Shell sort:

h	72	73	86	55	52	13	6	15	32	53	26	75	12	93	46	35	92	33	66
13	72	46	35	55	33	13	6	15	32	53	26	75	12	93	73	86	92	52	66
4	12	13	6	15	32	46	26	55	33	52	35	75	72	53	66	86	92	93	73
1	6	12	13	15	26	32	33	35	46	52	53	55	66	72	73	75	86	92	93

Chapter 10

Recursion

Text

1. Using unfolding solve the following recurrence equation

$$T(n) = \begin{cases} 3T(n/2) + n, & n \geq 2 \\ 1, & n = 1 \end{cases}$$

2. What is the aim of the following piece of code. Evaluate the complexity.

```
int max(inta[],intl,intr){
    int u, v;
    int m = (l + r)/2;
    if (l == r)
        return a[l];
    u = max (a, l, m);
    v = max (a, m+1, r);
    if (u > v)
        return u;
    else
        return v;
}
```

3. Recursively find the minimum number in an array.

4. Sort the following numbers using merge sort:

   38, 28, 43, 3, 9, 8, 10

5. Sort the following numbers using quick sort (always select the rightmost element as the pivot):

   7 28 1 50 87 68 21 10 67 8 41 70 47 48 61 30

6. Sort the following numbers using quick sort (always select the leftmost element as the pivot):

   13 15 2 7 4 8 9 12

7. Build a heap with the following array:

   5 4 9 7 19 8 17 2 6 5 21

8. Run heap sort on the following array:

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

9. Suppose to have an initially empty priority queue implemented with a heap. Given the following sequence of integers and * character:

   10 22 29 71 * * 58 * 34 9 31 * * 14 * 41 27 18 (1→48)

   where each integer corresponds to an insertion into the priority queue and character * corresponds to an extraction of the maximum. (p→v) asks you to change the item at position p into the new value v. Show the priority queue after each operation and return the sequence of values extracted.

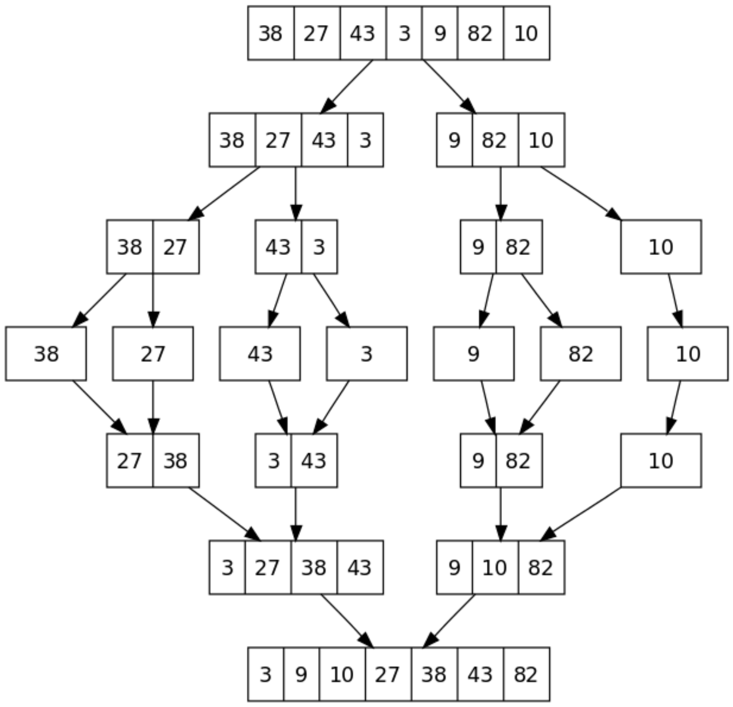
Solution

1. $O(n^{\log_2 3})$

2. $T(n) = 2T(n/2) + c$, which implies that $T(n) = O(n)$

3. Similar to the code of previous problem.

4. Merge sort:



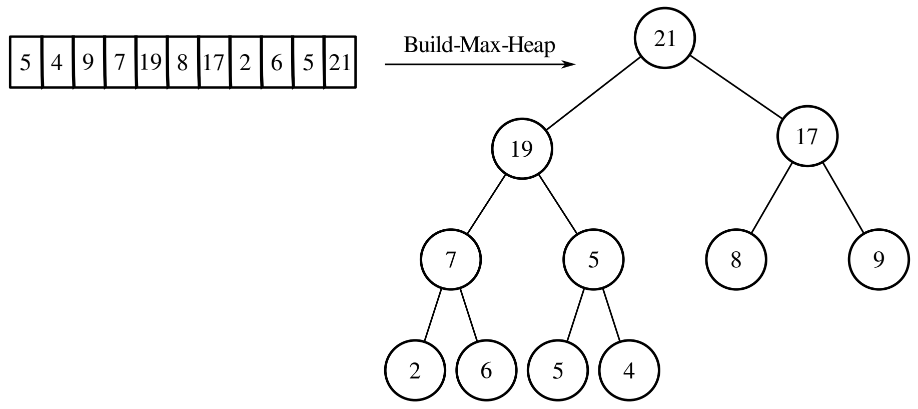
5. Quick sort:

7	28	1	50	87	68	21	10	67	8	41	70	47	48	61	30
7	28	1	21	10	8	30	87	67	68	41	70	47	48	61	50
7	1	8	21	10	28	30	87	67	68	41	70	47	48	61	50
1	7	8	21	10	28	30	87	67	68	41	70	47	48	61	50
1	7	8	21	10	28	30	87	67	68	41	70	47	48	61	50
1	7	8	10	21	28	30	87	67	68	41	70	47	48	61	50
1	7	8	10	21	28	30	41	47	48	50	70	67	68	61	87
1	7	8	10	21	28	30	41	47	48	50	70	67	68	61	87
1	7	8	10	21	28	30	41	47	48	50	70	67	68	61	87
1	7	8	10	21	28	30	41	47	48	50	70	67	68	61	87
1	7	8	10	21	28	30	41	47	48	50	61	67	68	70	87
1	7	8	10	21	28	30	41	47	48	50	61	67	68	70	87
1	7	8	10	21	28	30	41	47	48	50	61	67	68	70	87

6. Quick sort (leftmost):

12	2	7	4	8	9	**13**	15
9	2	7	4	8	**12**	**13**	15
8	2	7	4	**9**	**12**	**13**	15
4	2	7	**8**	**9**	**12**	**13**	15
4	2	**7**	**8**	**9**	**12**	**13**	15
2	**4**	**7**	**8**	**9**	**12**	**13**	15
2	**4**	**7**	**8**	**9**	**12**	**13**	15
2	**4**	**7**	**8**	**9**	**12**	**13**	**15**

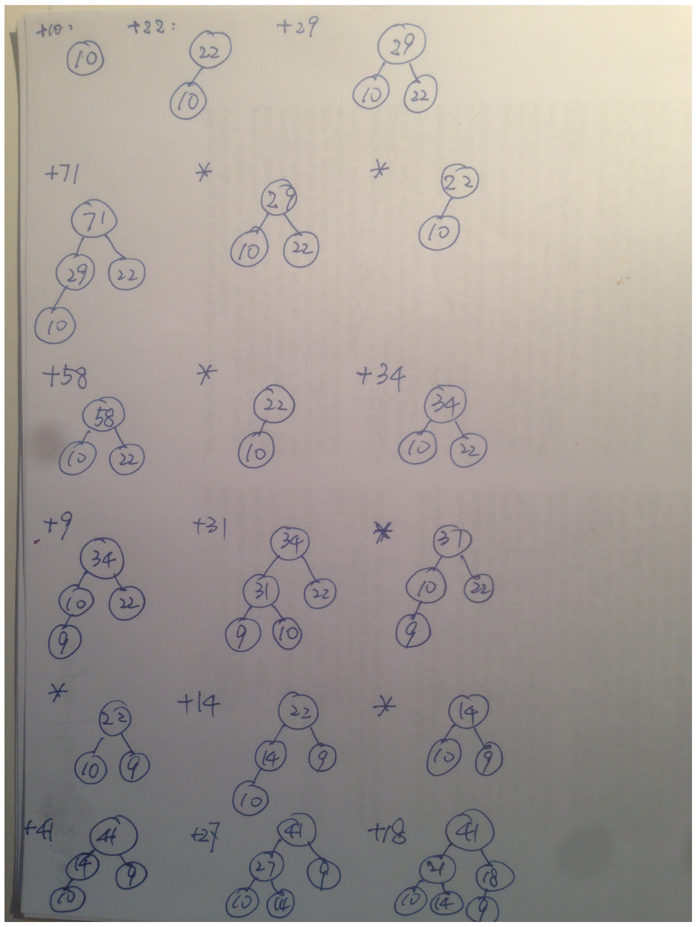
7. Build heap:



8. Heap sort: (**you must write in tree presentation**)

9	8	5	7	4	1	2	3	6	2
8	7	5	6	4	1	2	3	2	9
7	6	5	3	4	1	2	2	8	9
6	4	5	3	2	1	2	7	8	9
5	4	2	3	2	1	6	7	8	9
4	3	2	1	2	5	6	7	8	9
3	2	2	1	4	5	6	7	8	9
2	1	2	3	4	5	6	7	8	9
2	1	2	3	4	5	6	7	8	9
1	2	2	3	4	5	6	7	8	9
1	2	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1
1	2	2	3	4	5	6	7	8	9

9. Priority queue:



Chapter 11

Greedy Algorithm

Text

1. Using a greedy algorithm, evaluate an optimum Huffman code for the following sequence:

   a: 26, b: 6, c: 4, d: 3, e: 22, f: 13, g: 14, h: 9, i: 18, j: 10

Solution

1. Please draw the binary tree by yourself.

a	b	c	d	e
00	0110	01111	01110	111
f	g	h	i	j
010	100	1100	101	1101

Chapter 12

Data Structure

Text

1. Insert the sequence of integer keys LOVEONTOP in a hash-table of size 23. Consider each character corresponding to its progressive order in the alphabet, i.e., A=1, . . ., Z=26. Suppose the hash-table initially empty, and that open addressing with double hashing, with the two functions $h_1(k) = k \bmod 23$, $h_2(k) = 1 + (k \bmod 22)$, is used.

Solution

1. See the solution of the exam 2012.02.02.