Data Structure and Algorithm

Homework #5

=== Homework Reference Solution ===

Problem 1.

```
#include<stdio.h>
#include<stdlib.h>
/*************
We can treat the input as a graph where nodes are currency type and edges are exchange rates. A
matrix "R" can store these information. Besides, a three dimension array "back" is used to stored the
path and a three dimension array "P" is used to stored profit.
*******************/
double R[60][60];
double P[60][60][60];//[st_idx][k][C]
int back[60][60][60];//[st_idx][k][C]
int main()
{
	int n;
	int i,j,k;
	int a,b;
	double tmp;
	int find;
	int p;
	int t,s;
	int c[60];

	while(scanf("%d",&n)!=EOF)
	{
		/*************
		Read input data, and store them in R[i][j]
		***************/

		for(i=1;i<=n;++i)
			for(j=1;j<=n;++j)
				scanf("%lf",&R[i][j]);

		for(i=1;i<=n;++i) //start index
```

```
{
        P[i][0][i]=1;
        for(j=i+1;j<=n;++j)
                P[i][0][j]=0;
}
find=0;
/*****************
k is the number of steps. Since we want fewest exchange steps, we start by k = 1.
Iteratively check whether there exists a profitable sequence. The loop stops when a
profitable sequence is found
********************/
for(k=1;k<=n&&!find;++k) //#edges
{
        for(i=1;i<=n;++i) //start index
        {
                for(j=i;j<=n;++j)
                        P[i][k][j]=P[i][k-1][j];
                for(a=i;a<=n;++a)
                        for(b=i;b<=n;++b)
                        {
                                /*********************
                                For every path i->...->a within k-1 step, we check the profit of
                                adding one more node b at the end, i.e. i->...->a->b (k steps). If the
                                profit is higher, we record this profit and add node a in the back
                                tracking array "back".
                                *****************/
                                if((tmp=P[i][k-1][a]*R[a][b]) > P[i][k][b])
                                {
                                        P[i][k][b]=tmp;
                                        back[i][k][b]=a;
                                }
                        }

                /*****************
                Check whether the profit is more than 0.1. If yes, break the loop.
                *******************/
                if(P[i][k][i]>1.01)
                {
```

```
                    t=0;s=k;
                    p=i;
                    while(s>0)
                    {
                        c[++t]=p;
                        p=back[i][s--][p];
                    }
                    printf("%d",i);
                    while(t)
                        printf(",%d",c[t--]);
                    putchar('\n');
                    find=1;
                    break;
                }
                /*printf("start index=%d:\n",i);
                for(j=1;j<=n;++j)
                    printf("P[%d][%d][%d]=%lf\n",i,k,j,P[i][k][j]);
                putchar('\n');*/
            }

        }
        if(!find)
            puts("Not profitable");
    }

    return 0;
}
```

Problem 2.

(1)

Selection	A	B	C	D	E	F	G	H
A	0	3	∞	1	∞	∞	∞	∞
D	0	2	∞	1	∞	3	∞	∞
B	0	2	8	1	∞	3	∞	10
F	0	2	8	1	7	3	∞	10
E	0	2	8	1	7	3	12	10
C	0	2	8	1	7	3	12	10

H	0	2	8	1	7	3	12	10
G	0	2	8	1	7	3	12	10

(2)

k	b	c	d	e	f	g	h
1	∞	9	∞	10	∞	8	∞
2	15	9	12	3	9	8	15
3	8	9	5	3	2	8	12
4	8	9	5	2	2	8	5
5	7	9	4	2	1	8	5
6	7	9	4	1	1	8	4
7	6	9	3	1	0	8	4

(3)

No

The sub-path from simple paths may not be able to construct a simple path. For example, suppose the longest simple path from node 1 to node 2 is 1->4->2, and he longest simple path from node 2 to node 3 is 2->4->3. Obviously, 1->4->2->4->3 is not a simple path.

(4)

No

A longest path may not be composed of edges from longest path tree even if the weight of all edges are positive. d(u,w) + d(w,v) > d(u,v) does not imply that d(u,w) > d(u,v) and d(w,v) > d(u,v).

Problem 3.

(1)

We can run DFS algorithm on the graph. If we find a back edge on the graph, it must contain some cycles. Due to the fact that there exists a cycle in if we find |V| edges in the graph, the algorithm can be run in O(|V|).

(2)

We can run Bellman-Ford algorithm on the graph. After updating dist[] |V| times, we can determine whether there exists a negative cycle. If we want to find the vertices of the negative cycle, we need to keep the parent of the vertex on the shortest path every time. Therefore, if there exists a negative cycle in the graph, we just back-trace the parent of the updated vertex at V-th time, then we will find a negative cycle in the back-trace step.

Problem 4.

(1)

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

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

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

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

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

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

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

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

(1, 2, 3, 4, 5, 6, 7, 8, 9) → finished

(2)

	Unsorted		Sorted
	Left part		Pivot
	Right part		

Initial:

5	3	2	7	8	6	1	9	4

Pivot 5:

1	3	2	4	5	6	8	9	7

Pivot 1:

1	3	2	4	5	6	8	9	7

Pivot 3:

1	2	3	4	5	6	8	9	7

Pivot 2:

1	2	3	4	5	6	8	9	7

Pivot 4:

1	2	3	4	5	6	8	9	7

Pivot 6:

1	2	3	4	5	6	8	9	7

Pivot 8:

1	2	3	4	5	6	7	8	9

Pivot 7:

1	2	3	4	5	6	7	8	9

Pivot 9:

1	2	3	4	5	6	7	8	9

(3)

Sort a sorted list, e.g. (1, 2, 3, 4, 5, 6, 7, 8, 9), and always choose the left-most element as the pivot.

Time complexity = $(N - 1) + (N - 2) + \ldots + 1 = O(N^2)$

(4)

(((((5) (3)) (2)) ((7) (8))) (((6) (1)) ((9) (4))))

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

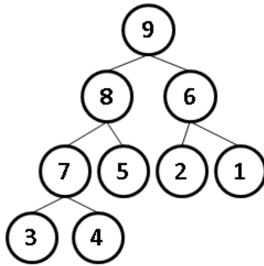
(2, 3, 5) (7, 8) (1, 6) (4, 9)

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

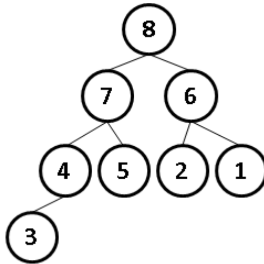
(1, 2, 3, 4, 5, 6, 7, 8, 9)

(5)

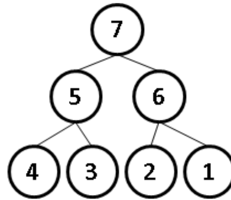
The max-heap after inserting all the elements:



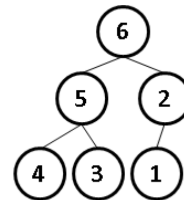
After extracting 9:



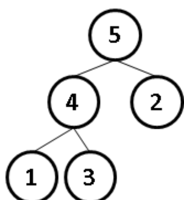
After extracting 8:



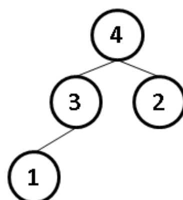
After extracting 7:



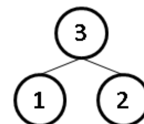
After extracting 6:



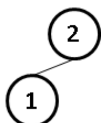
After extracting 5:



After extracting 4:



After extracting 3:



After extracting 2:

After extracting 1:

The heap is empty and the sorting is finished.

(6)

Before entering the fourth for loop:

B									
C	1	2	3	4	5	6	7	8	9

1st iteration:

B				4					
C	1	2	3	3	5	6	7	8	9

2nd iteration:

B				4					9
C	1	2	3	3	5	6	7	8	8

3rd iteration:

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

4th iteration:

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

5th iteration:

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

6th iteration:

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

7th iteration:

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

8th iteration:

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

9th iteration:

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

(7)

Initial:

501	939	1137	2345	666	34	218

1st pass:

501	34	2345	666	1137	218	939

2nd pass:

501	218	34	1137	939	2345	666

3rd pass:

34	1137	218	2345	501	666	939

4th pass:

34	218	501	666	939	1137	2345

(8)

The time complexity of Radix Sort Algorithm: $O((n + r)log_r k)$

The time complexity of Counting Sort Algorithm: $O(n + k)$

In this problem, $n = 7$, $k = 2345$, and $r = 10$, thus, the total computational cost of Counting Sort is much higher than Radix Sort.