DS

Sort

	平均复杂度	最优复杂度	最坏复杂度	辅助空间	稳定性
Bubble Sort	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	Stable
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Unstable
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	Stable
Shell Sort	$O(n^{\frac{7}{6}})$	$O(n)$	$O(n^{\frac{4}{3}})$	$O(1)$	Unstable
Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$	Unstable
Merge Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	Stable
Quick Sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$	Unstable
Bucket Sort	$O(n+\frac{n^2}{k}+k)$	$O(n)$	$O(n^2)$	$O(k+n)$	Stable

通过交换相邻元素的排序算法需要$\Omega(n^2)$

Quick Sort 的辅助空间是递归额外使用的栈空间。

对于归并排序的总体复杂度有:

$$T(N)=2T(N/2)+N$$

求解后:

$$T(N)=N+N\log N=O(N\log N)$$

我们总共需要的Merge Times 是$O(\log N)$, 每次Merge的时间开销大概是$O(\log N)$

杂项

斐波那契数列：递归：$O(2^n)$ 迭代：$O(n)$, 算法：二叉树叶子数就是运行时间，等价于求解$T(n)=T(n-1)+T(n-2)$

Tree

Nodes = Degree + 1

Graph

拓扑排序

https://blog.csdn.net/qq_41713256/article/details/80805338

邻接矩阵适合稠密图，邻接表适合稀疏图

$$\text{邻接矩阵复杂度：}O(V^2)$$
$$\text{邻接表的复杂度：}O(|V|+|E|)$$

1. 先统计所有节点的入度，对于入度为0的节点就可以分离出来，然后把这个节点指向的节点的入度减一。
2. 一直做这个操作，直到所有的节点都被分离出来。
3. 如果最后不存在入度为0的节点，那就说明有环，不存在拓扑排序，也就是很多题目的无解的情况。

有些拓扑排序要求字典序最小什么的，那就把队列换成优先队列就好了。

最短路径算法

Dijkstra

https://www.cnblogs.com/Glacier-elk/p/9438077.html

1. 遍历没有在最短路中的点，选出一个距离 **已经在最短路集合中的点** 距离最近的点。
2. 把它加入到最短路中。
3. 更新所有点的最短路。
4. 重复，直到所有的点都加入到最短路中。

```
void Dijkstra(Table T)
{
    Vertex V,W;
    while(1)
    {
        V = Smallest unknown distance vertex;
        if(V == NULL)
            break;
        T[V].Known = True;
        for each W adjacent to V
            if(!T[W].Known)
                if(T[V].Dist + Cvw < T[W].Dist)
                {
                    T[W].Dist = T[V].Dist + Cvw;
                    T[W].Path = V;
                }
    }
}
```

$$\text{复杂度：}O(V^2)$$

网络流问题

最小生成树

https://www.cnblogs.com/JoshuaMK/p/prim_kruskal.html

Prim算法适用于稠密图,Kruskal适用于稀疏图

Kruskal

Kruskal算法的步骤包括：

1. 对所有权值进行从小到大排序

2. 然后每次选取最小的权值，如果和已有点集构成环则跳过，否则加到该点集中。最终有所有的点集构成的树就是最佳的。

使用邻接表存储，并查集操作。

$$\text{复杂度}:O(E\log E)$$

Prim

Prim算法求最小生成树的时候和边数无关，和顶点树有关，所以适合求解稠密网的最小生成树。

Prim算法的步骤包括：

1. 将一个图分为两部分，一部分归为点集U，一部分归为点集V，U的初始集合为{V1}，V的初始集合为{ALL-V1}。
2. 针对U开始找U中各节点的所有关联的边的权值最小的那个，然后将关联的节点Vi加入到U中，并且从V中删除（注意不能形成环）。
3. 递归执行步骤2，直到V中的集合为空。
4. U中所有节点构成的树就是最小生成树。

使用邻接矩阵存储。

$$\text{复杂度}:O(V^2)$$

Hash Table

Hash Table Find的平均长度

处理办法	查找成功	查找失败	插入
Separate Chaining	$1+\frac{\lambda}{2}$	$\lambda+e^{-\lambda}$	与失败相同
Linear Probing	$\frac{1}{2}(1+\frac{1}{(1-\lambda)})$	$\frac{1}{2}(1+\frac{1}{(1-\lambda)^2})$	与失败相同
Quadratic Probing	$-\frac{1}{\lambda}\ln(1-\lambda)$	$\frac{1}{1-\lambda}$	与失败相同
Double Hashing	$-\frac{1}{\lambda}\ln(1-\lambda)$	$\frac{1}{1-\lambda}$	失败相同

Hash表的平均查找与插入时间复杂度为$O(1)$，但真正的复杂度取决于他的冲突解决方法，最坏复杂度为$O(n)$

查找失败总是比查找成功的平均长度长

定理：如果hash table的长度是质数，且表至少有一半是空的，那么使用Quadratic probing 总是能插入一个新的元素。

如果这个素数的形式是4k+3，且使用probing方法是$F(i)=\pm i^2$，那么整个哈希表都可以被探测到。

$F(i)=\pm i^2$的含义是每次探测先加再减

程序填空题

2015-2016秋冬期末

5-1 Please fill the array with the results after the following union/find operations.

```
union( find(4), find(6) )
union( find(2), find(7) )
union( find(0), find(4) )
union( find(7), find(6) )
union( find(7), find(1) )
```

Note: Assume `union-by-size` (if two sets are equal-sized, the first root will be the root of the result) and `find-with-path-compression`. `S[i]` is initialized to be -1 for all $0 \le i \le 7$.

i	0	1	2	3	4	5	6	7
S[i]	4	-6 (1分)	6 (1分)	-1	1 (1分)	-1	4	2 (1分)
		4	4		-6			4

评测结果：答案错误（0 分）

序号	结果	得分
0	答案错误	0
1	答案错误	0
2	答案错误	0
3	答案错误	0

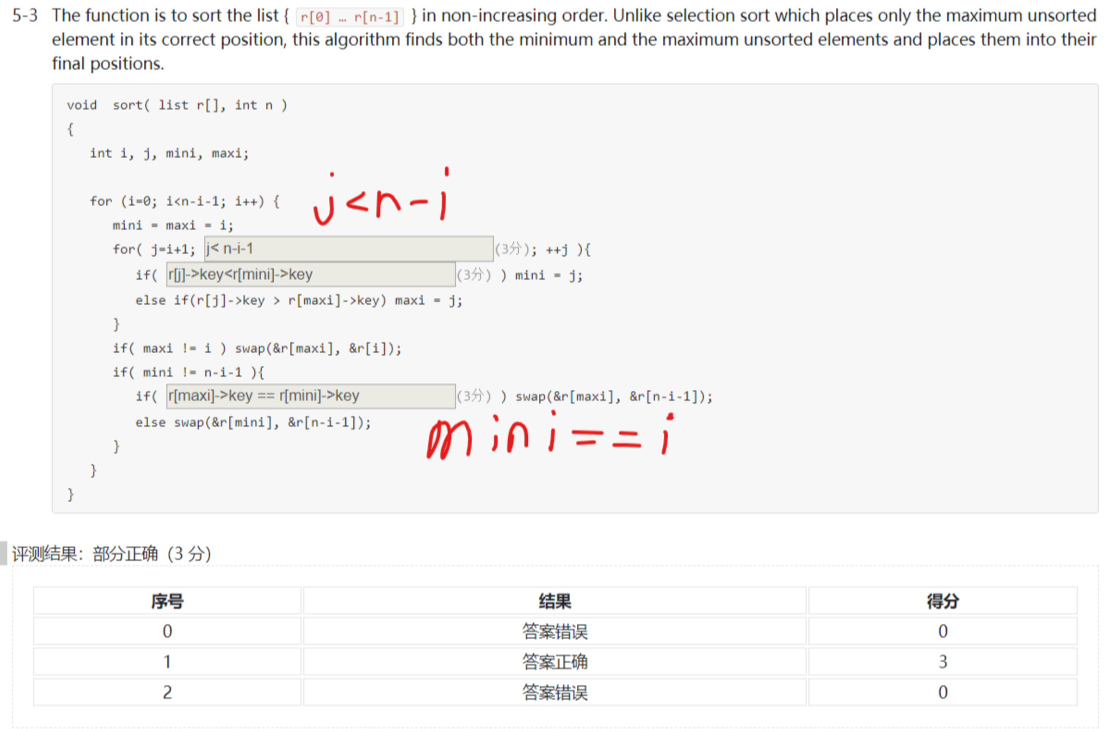
5-2 A binary tree is said to be "height balanced" if both its left and right subtrees are height balanced, and the heights of its left and right subtrees can differ by at most 1. That is, $|H_L - H_R| \le 1$ where H_L and H_R are the heights of the left and right subtrees, respectively. An empty binary tree is defined to be height balanced.

The function `IsBalanced` is to judge if a given binary tree `T` is height balanced. If the answer is yes then return `true` and store the tree height in the parameter `pHeight`, else simply return `false`. The height of an empty tree is defined to be 0.

```
typedef struct TNode *BinTree;
struct TNode{
    int Key;
    BinTree Left;
    BinTree Right;
};

bool IsBalanced ( BinTree T, int *pHeight )
{
    int  LHeight, RHeight, diff;

    if( T == NULL) {
        *pHeight = 0;
        return true;
    }
    else if ( IsBalanced(T->Left, &LHeight) && IsBalanced(T->Right, &RHeight) ) {
        diff = LHeight - RHeight;
        if ( (diff<=1)&&(diff>=-1) (6分) )  {
            *pHeight = 1 + ( diff<0 ? -1*diff:diff (6分) );
            return true;        ( RH : LH )
        }
        else return false;
    }
    return false;
}
```

评测结果：部分正确（6 分）

序号	结果	得分
0	答案正确	6
1	答案错误	0



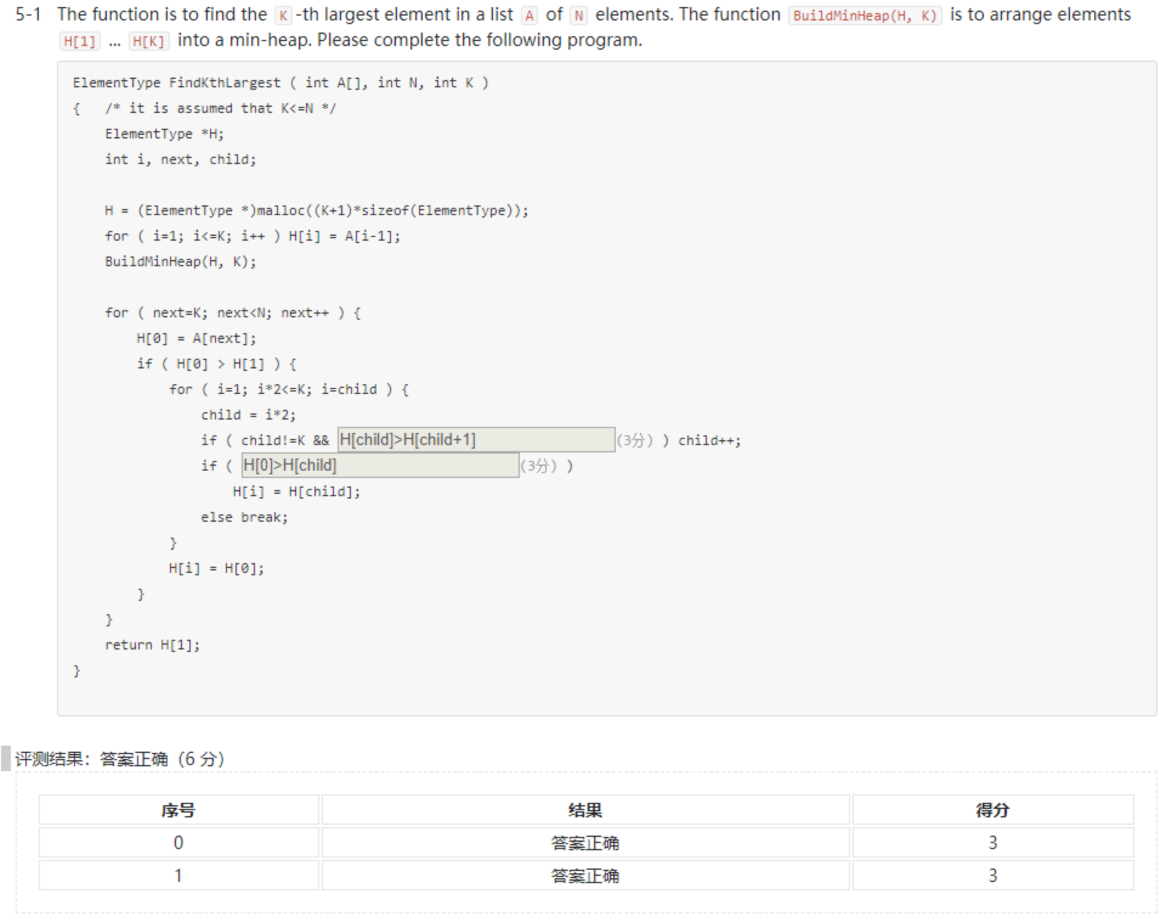
5-3 The function is to sort the list { `r[0] … r[n-1]` } in non-increasing order. Unlike selection sort which places only the maximum unsorted element in its correct position, this algorithm finds both the minimum and the maximum unsorted elements and places them into their final positions.

```
void  sort( list r[], int n )
{
   int i, j, mini, maxi;

   for (i=0; i<n-i-1; i++) {        j<n-i
      mini = maxi = i;
      for( j=i+1; j< n-i-1 (3分); ++j ){
         if( r[j]->key<r[mini]->key (3分) ) mini = j;
         else if(r[j]->key > r[maxi]->key) maxi = j;
      }
      if( maxi != i ) swap(&r[maxi], &r[i]);
      if( mini != n-i-1 ){
         if( r[maxi]->key == r[mini]->key (3分) ) swap(&r[maxi], &r[n-i-1]);
         else swap(&r[mini], &r[n-i-1]);        mini==i
      }
   }
}
```

评测结果：部分正确（3 分）

序号	结果	得分
0	答案错误	0
1	答案正确	3
2	答案错误	0

2016-2017秋冬期末

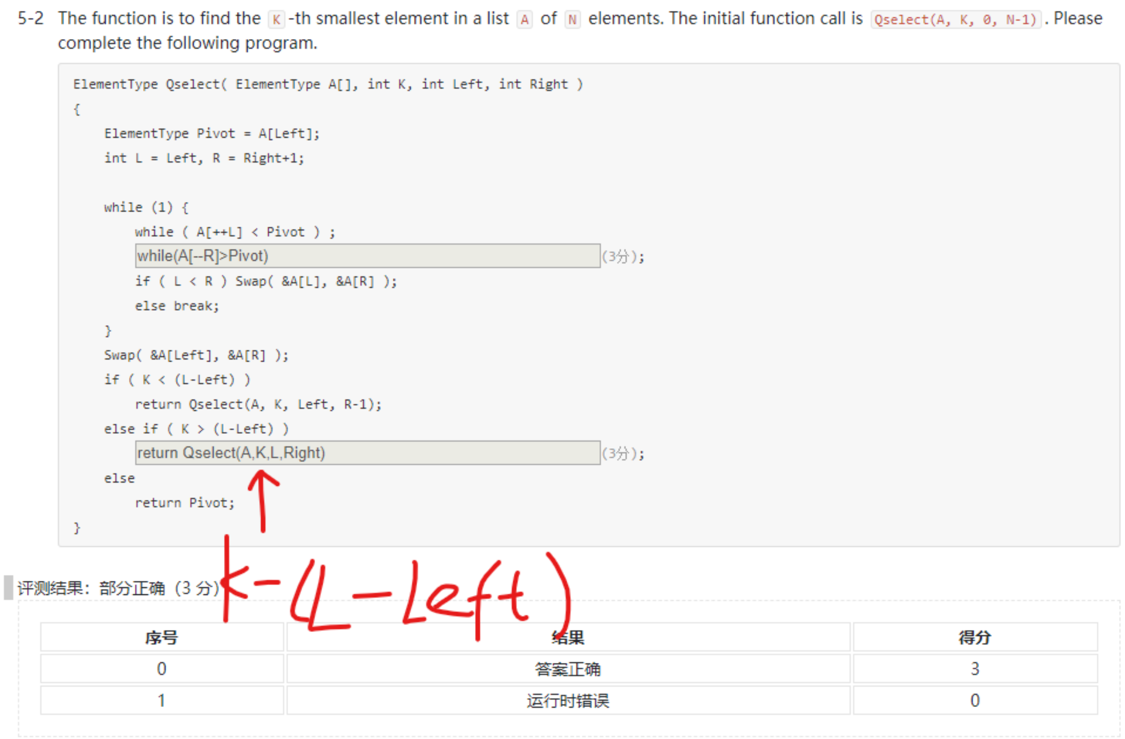


5-1 The function is to find the `K` -th largest element in a list `A` of `N` elements. The function `BuildMinHeap(H, K)` is to arrange elements `H[1]` … `H[K]` into a min-heap. Please complete the following program.

```
ElementType FindKthLargest ( int A[], int N, int K )
{   /* it is assumed that K<=N */
    ElementType *H;
    int i, next, child;

    H = (ElementType *)malloc((K+1)*sizeof(ElementType));
    for ( i=1; i<=K; i++ ) H[i] = A[i-1];
    BuildMinHeap(H, K);

    for ( next=K; next<N; next++ ) {
        H[0] = A[next];
        if ( H[0] > H[1] ) {
            for ( i=1; i*2<=K; i=child ) {
                child = i*2;
                if ( child!=K && H[child]>H[child+1] (3分) ) child++;
                if ( H[0]>H[child] (3分) )
                    H[i] = H[child];
                else break;
            }
            H[i] = H[0];
        }
    }
    return H[1];
}
```

评测结果：答案正确（6 分）

序号	结果	得分
0	答案正确	3
1	答案正确	3



5-2 The function is to find the `K`-th smallest element in a list `A` of `N` elements. The initial function call is `Qselect(A, K, 0, N-1)`. Please complete the following program.

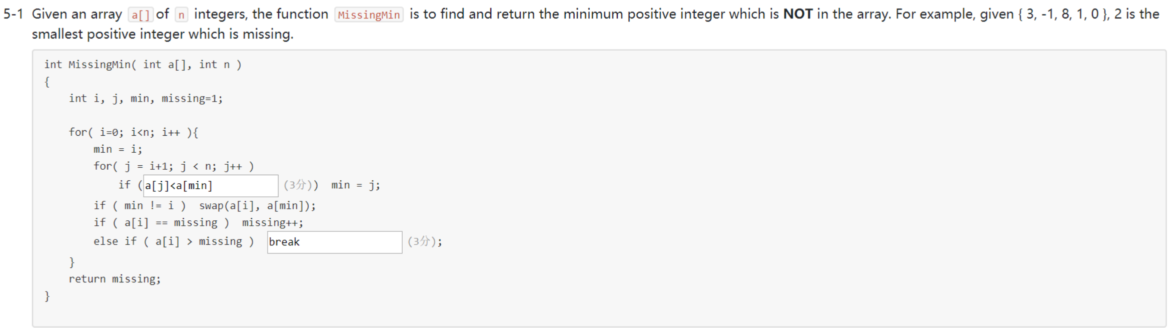
```
ElementType Qselect( ElementType A[], int K, int Left, int Right )
{
    ElementType Pivot = A[Left];
    int L = Left, R = Right+1;

    while (1) {
        while ( A[++L] < Pivot ) ;
        while(A[--R]>Pivot) (3分);
        if ( L < R ) Swap( &A[L], &A[R] );
        else break;
    }
    Swap( &A[Left], &A[R] );
    if ( K < (L-Left) )
        return Qselect(A, K, Left, R-1);
    else if ( K > (L-Left) )
        return Qselect(A,K,L,Right) (3分);
    else
        return Pivot;
}
```

k-(L-Left)

评测结果：部分正确（3 分）

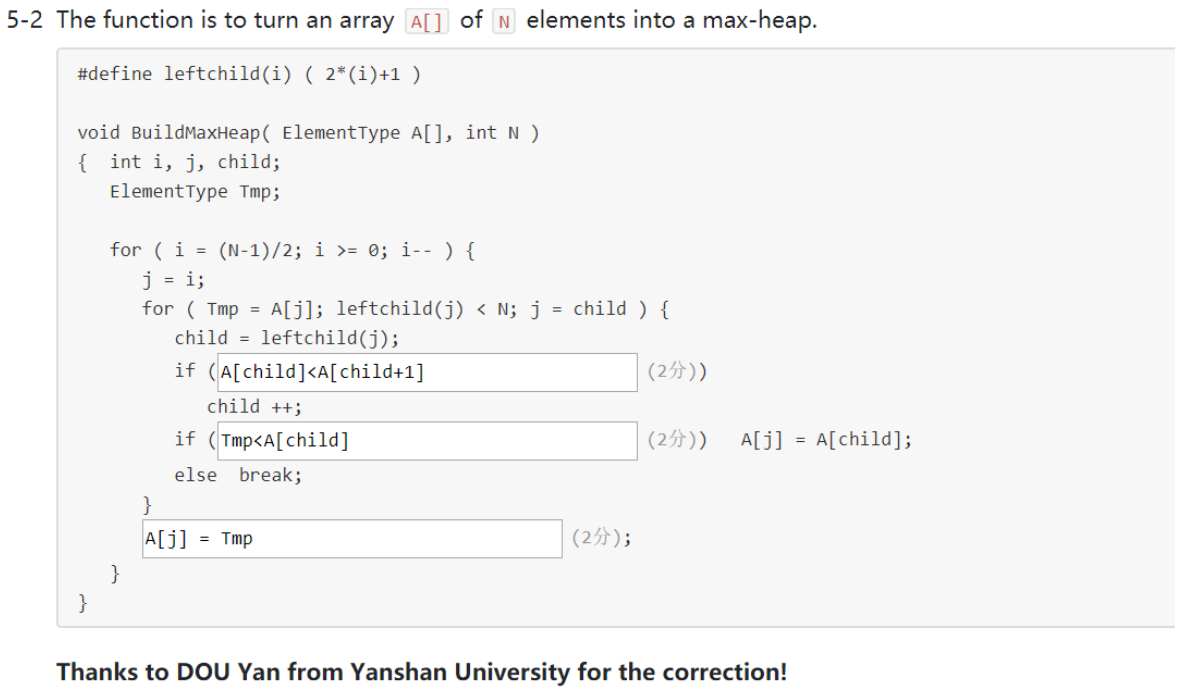
序号	结果	得分
0	答案正确	3
1	运行时错误	0

2017-18秋冬期末



5-1 Given an array `a[]` of `n` integers, the function `MissingMin` is to find and return the minimum positive integer which is **NOT** in the array. For example, given { 3, -1, 8, 1, 0 }, 2 is the smallest positive integer which is missing.

```
int MissingMin( int a[], int n )
{
    int i, j, min, missing=1;

    for( i=0; i<n; i++ ){
        min = i;
        for( j = i+1; j < n; j++ )
            if (a[j]<a[min]          (3分))  min = j;
        if ( min != i )  swap(a[i], a[min]);
        if ( a[i] == missing )  missing++;
        else if ( a[i] > missing )  break          (3分);
    }
    return missing;
}
```



5-2 The function is to turn an array `A[]` of `N` elements into a max-heap.

```
#define leftchild(i) ( 2*(i)+1 )

void BuildMaxHeap( ElementType A[], int N )
{  int i, j, child;
   ElementType Tmp;

   for ( i = (N-1)/2; i >= 0; i-- ) {
      j = i;
      for ( Tmp = A[j]; leftchild(j) < N; j = child ) {
         child = leftchild(j);
         if (A[child]<A[child+1]          (2分))
            child ++;
         if (Tmp<A[child]          (2分))   A[j] = A[child];
         else  break;
      }
      A[j] = Tmp          (2分);
   }
}
```

Thanks to DOU Yan from Yanshan University for the correction!

2019-2019秋冬期末

5-1 The function is to find the `K`-th smallest element in a list `A` of `N` elements. The function `BuildMaxHeap(H, K)` is to arrange elements `H[1]` ... `H[K]` into a max-heap. Please complete the following program.

```
ElementType FindKthSmallest ( int A[], int N, int K )
{   /* it is assumed that K<=N */
    ElementType *H;
    int i, next, child;

    H = (ElementType *)malloc((K+1)*sizeof(ElementType));
    for ( i=1; i<=K; i++ ) H[i] = A[i-1];
    BuildMaxHeap(H, K);

    for ( next=K; next<N; next++ ) {
        H[0] = A[next];
        if ( H[0] < H[1] ) {
            for ( i=1; i*2<=K; i=child ) {
                child = i*2;
                if ( child!=K && H[child] < H[child+1](3分) ) child++;
                if ( H[0] < H[child](3分) )
                    H[i] = H[child];
                else break;
            }
            H[i] = H[0];
        }
    }
    return H[1];
}
```

评测结果：答案正确（6 分）

序号	结果	得分
0	答案正确	3
1	答案正确	3

5-2 The function is to find the `K`-th largest element in a list `A` of `N` elements. The initial function call is `Qselect(A, K, 0, N-1)`. Please complete the following program.

```
ElementType Qselect( ElementType A[], int K, int Left, int Right )
{
    ElementType Pivot = A[Left];
    int L = Left, R = Right+1;

    while (1) {
        while ( A[++L] > Pivot ) ;
        while ( A[--R] < Pivot )(3分);
        if ( L < R ) Swap( &A[L], &A[R] );
        else break;
    }
    Swap( &A[Left], &A[R] );
    if ( K < (L-Left) )
        return Qselect(A, K, Left, R-1);
    else if ( K > (L-Left) )
        return Qselect(A, K-(L-Left), L, Right)(3分);
    else
        return Pivot;
}
```

评测结果：答案正确（6 分）

序号	结果	得分
0	答案正确	3
1	答案正确	3

函数题

2015-2016秋冬期末

6-1 CheckBST[2]

Given a binary tree, you are supposed to tell if it is a binary search tree. If the answer is yes, try to find the K-th smallest key, else try to find the height of the tree.

Format of function:

```
int CheckBST ( BinTree T, int K );
```

where `BinTree` is defined as the following:

```
typedef struct TNode *BinTree;
struct TNode{
    int Key;
    BinTree Left;
    BinTree Right;
};
```

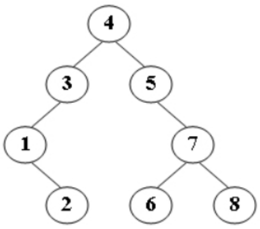
The function `CheckBST` is supposed to return the `K`-th smallest key if `T` is a binary search tree; or if not, return the negative height of `T` (for example, if the height is 5, you must return -5).

Here the height of a leaf node is defined to be 1. `T` is not empty and all its keys are positive integers. `K` is positive and is never more than the total number of nodes in the tree.

Sample program of judge:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct TNode *BinTree;
struct TNode{
    int Key;
    BinTree Left;
    BinTree Right;
};

BinTree BuildTree(); /* details omitted */
int CheckBST ( BinTree T, int K );

int main()
{
    BinTree T;
    int K, out;

    T = BuildTree();
    scanf("%d", &K);
    out = CheckBST(T, K);
    if ( out < 0 )
        printf("No.  Height = %d\n", -out);
    else
        printf("Yes.  Key = %d\n", out);

    return 0;
}
/* 你的代码将被嵌在这里 */
```

Sample Input 1: (for the following tree)

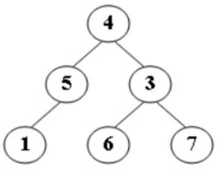


```
3
```

Sample Output 1:

```
Yes.  Key = 3
```

Sample Input 2: (for the following tree)



```
2
```

Sample Output 2:

```
No.  Height = 3
```

不想写了，写了几遍都看错，看成写判断平衡二叉树，艹。

思路应该挺简单的。

2016-2017秋冬期末

函数题

总分：8　得分：8

6-1 LCA in BST

The lowest common ancestor (LCA) of two nodes `u` and `v` in a tree `T` is the deepest node that has both `u` and `v` as descendants. Given any two nodes in a binary search tree (BST), you are supposed to find their LCA.

Format of function:

```
int LCA( Tree T, int u, int v );
```

where `Tree` is defined as the following:

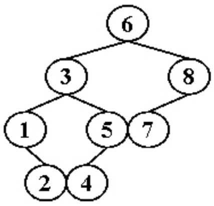
```
typedef struct TreeNode *Tree;
struct TreeNode {
    int   Key;
    Tree  Left;
    Tree  Right;
};
```

The function `LCA` is supposed to return the key value of the LCA of `u` and `v` in `T`. In case that either `u` or `v` is not found in `T`, return `ERROR` instead.

Sample program of judge:

```
#include <stdio.h>
#include <stdlib.h>

#define ERROR -1
typedef struct TreeNode *Tree;
struct TreeNode {
    int   Key;
    Tree  Left;
    Tree  Right;
};

Tree BuildTree(); /* details omitted */
int LCA( Tree T,  int u, int v );

int main()
{
    Tree T;
    int u, v, ans;

    T = BuildTree();
    scanf("%d %d", &u, &v);
    ans = LCA(T, u, v);
    if ( ans == ERROR ) printf("Wrong input\n");
    else printf("LCA = %d\n", ans);

    return 0;
}

/* Your function will be put here */
```

Sample Input 1 (for the tree shown in the Figure):



```
2 7
```

Sample Output 1:

```
LCA = 6
```

Sample Input 2 (for the same tree in Sample 1):

```
1 9
```

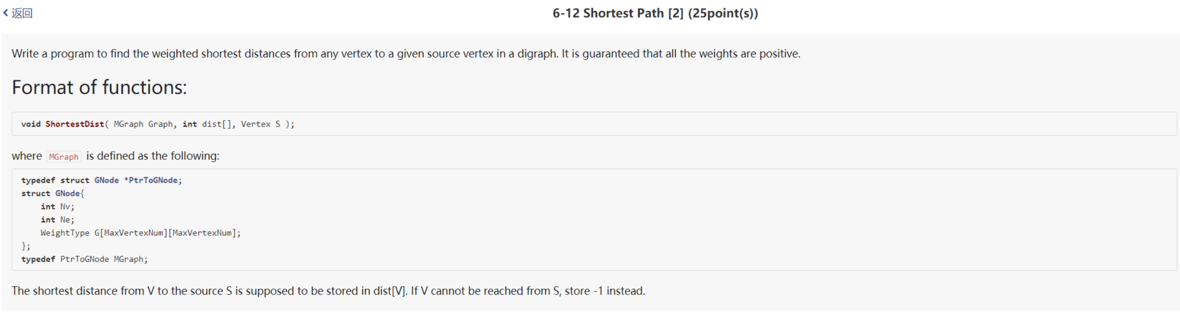
Sample Output 2:

```
Wrong input
```

AC代码

```
#define MAXN 128

int dfs(Tree T, int X, int *Stack, int *Sp);

int LCA(Tree T, int u, int v) {
    int stack1[MAXN];
    int stack2[MAXN];
    int sp1 = -1;
    int sp2 = -1;
    int f1 = dfs(T, u, stack1, &sp1);
    int f2 = dfs(T, v, stack2, &sp2);
    if (f1 * f2 == 0) {
        return ERROR;
    }
    int min = (sp1 - sp2 < 0) ? sp1 : sp2;
    for (int i = 0; i <min+1; ++i) {
        if (stack1[i]!=stack2[i])
        {
            return stack1[i-1];
        }
    }
    return stack1[min];


}

int dfs(Tree T, int X, int *Stack, int *Sp) {
    Tree cur = T;
    while (1) {
        if (cur == NULL) {
            return 0;
        }
        Stack[++*Sp] = cur->Key;
        if (cur->Key > X) {
            cur = cur->Left;
        } else if (cur->Key < X) {
            cur = cur->Right;
        } else {
            return 1;
        }
    }

}
```

2017-18秋冬期末



6-12 Shortest Path [2] (25point(s))

Write a program to find the weighted shortest distances from any vertex to a given source vertex in a digraph. It is guaranteed that all the weights are positive.

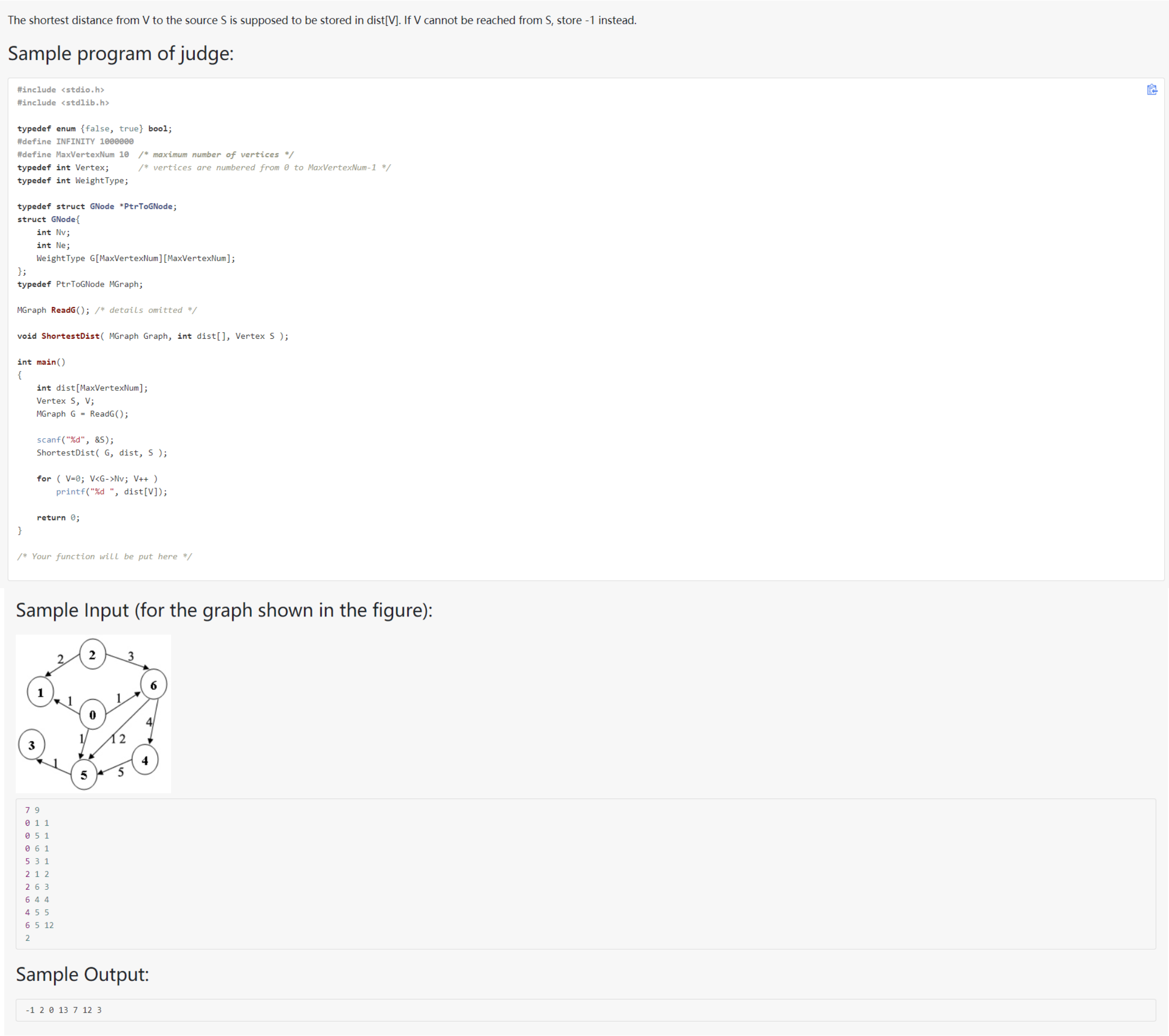
Format of functions:

```
void ShortestDist( MGraph Graph, int dist[], Vertex S );
```

where `MGraph` is defined as the following:

```
typedef struct GNode *PtrToGNode;
struct GNode{
    int Nv;
    int Ne;
    WeightType G[MaxVertexNum][MaxVertexNum];
};
typedef PtrToGNode MGraph;
```

The shortest distance from V to the source S is supposed to be stored in dist[V]. If V cannot be reached from S, store -1 instead.



The shortest distance from V to the source S is supposed to be stored in dist[V]. If V cannot be reached from S, store -1 instead.

Sample program of judge:

```c
#include <stdio.h>
#include <stdlib.h>

typedef enum {false, true} bool;
#define INFINITY 1000000
#define MaxVertexNum 10  /* maximum number of vertices */
typedef int Vertex;      /* vertices are numbered from 0 to MaxVertexNum-1 */
typedef int WeightType;

typedef struct GNode *PtrToGNode;
struct GNode{
    int Nv;
    int Ne;
    WeightType G[MaxVertexNum][MaxVertexNum];
};
typedef PtrToGNode MGraph;

MGraph ReadG(); /* details omitted */

void ShortestDist( MGraph Graph, int dist[], Vertex S );

int main()
{
    int dist[MaxVertexNum];
    Vertex S, V;
    MGraph G = ReadG();

    scanf("%d", &S);
    ShortestDist( G, dist, S );

    for ( V=0; V<G->Nv; V++ )
        printf("%d ", dist[V]);

    return 0;
}

/* Your function will be put here */
```

Sample Input (for the graph shown in the figure):

```
7 9
0 1 1
0 5 1
0 6 1
5 3 1
2 1 2
2 6 3
6 4 4
4 5 5
6 5 12
2
```

Sample Output:

```
-1 2 0 13 7 12 3
```

AC代码

```c
void ShortestDist(MGraph Graph, int dist[], Vertex S) {
    int vis[MaxVertexNum];
    int min_dist;
    int min_dist_v;
    for (int i = 0; i < Graph->Nv; ++i) {
        vis[i] = 0;
        dist[i] = INFINITY;
    }
    dist[S] = 0;

    while (1) {
        min_dist = -1;
        min_dist_v = INFINITY;
        for (int i = 0; i < Graph->Nv; ++i) {
            if (min_dist_v > dist[i] && !vis[i]) {
                min_dist_v = dist[i];
                min_dist = i;
            }
        }
        if (min_dist == -1) {
            break;
        }
        vis[min_dist] = 1;
        for (int i = 0; i < Graph->Nv; ++i) {
            if (Graph->G[min_dist][i] > 0 && Graph->G[min_dist][i] +
dist[min_dist] < dist[i]) {
```

```
                dist[i] = Graph->G[min_dist][i] + dist[min_dist];
            }
        }

    }

    for (int i = 0; i < Graph->Nv; ++i) {
        if (dist[i] == INFINITY)
            dist[i] = -1;
    }

}
```

2018-2019秋冬期末

6-1 Check Topological Order

Write a program to test if a give sequence `Seq` is a topological order of a given graph `G`.

Format of functions:

```
bool IsTopSeq( Vertex Seq[], LGraph G );
```

where `LGraph` is defined as the following:

```
typedef struct AdjVNode *PtrToAdjVNode;
struct AdjVNode{
    Vertex AdjV;
    PtrToAdjVNode Next;
};

typedef struct Vnode{
    PtrToAdjVNode FirstEdge;
} AdjList[MaxVertexNum];

typedef struct GNode *PtrToGNode;
struct GNode{
    int N_v;
    int N_e;
    AdjList G;
};
typedef PtrToGNode LGraph;
```

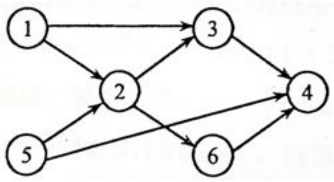
The function `IsTopSeq` must return `true` if `Seq` does correspond to a topological order; otherwise return `false`.

Note: Although the vertices are numbered from 1 to MaxVertexNum, they are **indexed from 0** in the LGraph structure.

Sample program of judge:

```
#include <stdio.h>
#include <stdlib.h>

typedef enum {false, true} bool;
#define MaxVertexNum 10  /* maximum number of vertices */
typedef int Vertex;      /* vertices are numbered from 1 to MaxVertexNum */

typedef struct AdjVNode *PtrToAdjVNode; 
struct AdjVNode{
    Vertex AdjV;
    PtrToAdjVNode Next;
};

typedef struct Vnode{
    PtrToAdjVNode FirstEdge;
} AdjList[MaxVertexNum];

typedef struct GNode *PtrToGNode;
struct GNode{  
    int N_v;
    int N_e;
    AdjList G;
};
typedef PtrToGNode LGraph;

LGraph ReadG(); /* details omitted */

bool IsTopSeq( Vertex Seq[], LGraph G );

int main()
{
    int i, j, N;
    Vertex Seq[MaxVertexNum];
    LGraph G = ReadG();
    scanf("%d", &N);
    for (i=0; i<N; i++) {
        for (j=0; j<G->N_v; j++)
            scanf("%d", &Seq[j]);
        if ( IsTopSeq(Seq, G)==true ) printf("yes\n");
        else printf("no\n");
    }

    return 0;
}

/* Your function will be put here */
```

Sample Input (for the graph shown in the figure):



```
6 8
1 2
1 3
5 2
5 4
2 3
2 6
3 4
6 4
5
1 5 2 3 6 4
5 1 2 6 3 4
5 1 2 3 6 4
5 2 1 6 3 4
1 2 3 4 5 6
```

Sample Output:

```
yes
yes
yes
no
no
```

AC代码

```
typedef struct Queue {
    Vertex Arr[128];
    int front;
    int rear;
} QueueADT;

typedef QueueADT *Queue;

Queue CreateQueue() {
    Queue Q = malloc(sizeof(QueueADT));
    Q->front = 0;
    Q->rear = -1;
    return Q;
}

void Enqueue(Queue Q, Vertex V) {
    Q->Arr[++Q->rear] = V;
}

Vertex Dequeue(Queue Q) {
    return Q->Arr[Q->front--];
}

int isEmpty(Queue Q) {
    return Q->rear + 1 == Q->front;
}

int isInQueue(Queue Q, Vertex X) {
```

```c
    for (int i = Q->front; i <= Q->rear; i++) {
        if (Q->Arr[i] == X)
            return 1;
    }
    return 0;
}

/* Your function will be put here */
bool IsTopSeq(Vertex Seq[], LGraph G) {
    Queue Q = CreateQueue();
    for (int i = 0; i < G->N_v; ++i) {
        int cur = Seq[i] - 1;
        for (int j = 0; j < G->N_v; ++j) {
            if (isInQueue(Q, j) || j == cur) {
                continue;
            }

            PtrToAdjVNode temp = G->G[j].FirstEdge;
            while (temp != NULL) {
                if (temp->AdjV == cur) {
                    return false;
                }
                temp = temp->Next;
            }
        }
        Enqueue(Q, cur);
    }
    return true;
}
```