Simulation Queue: typedef struct TNode{int value; struct TNode\* next;}Node;

Node\* head; Node\* tail;

Node\* makeNode(int v){Node\* p = (Node\*)malloc(sizeof(Node)); p->value = v; p->next = NULL; return p;}

int queueEmpty(){return head == NULL && tail == NULL;}

void push(int v){Node\* p = makeNode(v); if(queueEmpty()){head = p; tail = p; return;}tail->next = p; tail = p;}

int pop(){if(queueEmpty()){return -1;}Node\* tmp = head; int v = head->value; head = head->next;

if(head == NULL) tail = NULL; free(tmp); return v;}

int main(){head = NULL; tail = NULL;char cmd[50];

while(1){scanf("%s",cmd); if(strcmp(cmd,"#")==0){break;}

else if(strcmp(cmd,"PUSH")==0){int v; scanf("%d",&v); push(v);}

else if(strcmp(cmd,"POP")==0){if(queueEmpty()){printf("NULL\n");}else{int v = pop(); printf("%d\n",v);}}}

return 0;}

Check Pathensis: struct Stack{char data; struct StackNode\* next;};

struct Stack\* createStack(char data){struct Stack\* newNode = (struct Stack\*)malloc(sizeof(struct Stack));

newNode->data = data; newNode->next = NULL; return newNode;}

int isEmpty(struct Stack\* root) {return !root;}

void push(struct StackNode\*\* root, char data){struct Stack\* newNode = createStack(data);

newNode->next = \*root; \*root = newNode;}

char pop(struct Stack\*\* root){if (isEmpty(\*root)){return '\0';}

struct Stack\* temp = \*root; \*root = (\*root)->next; char popped = temp->data; free(temp); return popped;}

char peek(struct StackNode\* root){if (isEmpty(root)){return '\0';} return root->data;}

int isMatchingPair(char open, char close){

return ((open == '(' && close == ')') || (open == '[' && close == ']') || (open == '{' && close == '}'));}

int isCorrectExpression(char\* s){struct Stack\* stack = NULL;

for (int i = 0; i < strlen(s); i++){if (s[i] == '(' || s[i] == '[' || s[i] == '{') {push(&stack, s[i]);}

else if (s[i] == ')' || s[i] == ']' || s[i] == '}') {if (isEmpty(stack) || !isMatchingPair(pop(&stack), s[i])){return 0;}}}

return isEmpty(stack);}

Sorting Integer: #define N 100001 int a[N]; int n;

void swap(int i,int j){int t=a[i]; a[i]=a[j]; a[j]=t;}

void heapify(int i, int n){int L=2\*i; int R=2\*i+1; int maxIdx=i;

if(L<=n&&a[maxIdx]<a[L])maxIdx=L; if(R<=n&&a[maxIdx]<a[R])maxIdx=R;

if(maxIdx!=i){swap(i,maxIdx);heapify(maxIdx,n);}}

void buildHeap(){for(int i=n/2; i>=1; i--) heapify(i,n);}

void heapSort(){buildHeap(); for(int i=n;i>=2;i--){swap(1,i); heapify(1,i-1);}}

int main(){scanf("%d",&n); for(int i=1;i<=n;i++) scanf("%d",&a[i]); heapSort();

for(int i=1;i<=n;i++) printf("%d ",a[i]); printf("\n"); return 0;}

Sorting strings: #define N 100001 #define MAX\_LEN 101 char\* s[N]; int n;

void input(){scanf("%d",&n); char str[MAX\_LEN];

for(int i=1;i<=n;i++){scanf("%s",str); s[i]=(char\*)malloc(strlen(str)+1); strcpy(s[i], str);}}

void swap(int i,int j){char\* t=s[i]; s[i]=s[j]; s[j]=t;}

void heapify(int i,int n){int L=2\*i; int R=2\*i+1; int maxIdx=i;

if(L<=n&&strcmp(s[maxIdx],s[L])<0) maxIdx=L;if(R<=n&&strcmp(s[maxIdx],s[R])<0) maxIdx=R;

if(maxIdx!=i){swap(i,maxIdx); heapify(maxIdx,n);}}

void buildHeap(){for(int i=n/2; i>=1; i--){heapify(i,n);}}

void heapSort(){buildHeap(); for(int i=n; i>=2; i--){swap(1,i); heapify(1,i-1);}}

void freeMemory(){for(int i=1; i<=n; i++){free(s[i]);}}

int main(){input(); heapSort(); for(int i=1; i<=n; i++){printf("%s\n",s[i]);}freeMemory(); return 0;}

Ranking learning ability: #define N 100001

typedef struct Student{char ID[11]; int grade; int pos;}Student; Student S[N]; int n;

void input(){scanf("%d",&n); for(int i=1; i<=n; i++){scanf("%s %d",S[i].ID, &S[i].grade);}}

void swap(int i,int j){Student t=S[i]; S[i]=S[j]; S[j]=t;}

void heapify\_grade(int i,int n){int L=2\*i; int R=2\*i+1; int maxIdx=i;

if(L<=n&&S[maxIdx].grade<S[L].grade) maxIdx=L; if(R<=n&&S[maxIdx].grade<S[R].grade) maxIdx=R;

if(maxIdx!=i){swap(i,maxIdx); heapify\_grade(maxIdx,n);}}

void buildHeap\_grade(){for(int i=n/2; i>=1; i--){heapify\_grade(i,n);}}

void heapSort\_grade(){buildHeap\_grade(); for(int i=n; i>=2; i--){swap(1,i); heapify\_grade(1,i-1);}}

void heapify\_ID(int i,int n){int L=2\*i; int R=2\*i+1; int maxIdx=i; if(L<=n&&strcmp(S[maxIdx].ID,S[L].ID)<0) maxIdx=L;

if(R<=n&&strcmp(S[maxIdx].ID,S[R].ID)<0) maxIdx=R;

if(maxIdx!=i){swap(i,maxIdx); heapify\_ID(maxIdx,n);}}

void buildHeap\_ID(){for(int i=n/2; i>=1; i--){heapify\_ID(i,n);}}

void heapSort\_ID(){buildHeap\_ID(); for(int i=n; i>=2; i--){swap(1,i); heapify\_ID(1,i-1);}}

void print(){for(int i=1; i<=n; i++){printf("%s %d\n",S[i].ID,S[i].pos);}}

int main(){input(); heapSort\_grade(); for(int i=1; i<=n; i++){S[i].pos=i-1;}heapSort\_ID(); print(); return 0;}

Sorting vectors of integers: #define N 100001 int\* a[N]; int n,m;

void input(){scanf("%d%d",&n,&m);for(int i=1; i<=n; i++){a[i]=(int\*)malloc(sizeof(int)\*(m+1)); for(int j=1;j<=m;j++) scanf("%d",&a[i][j]);}}

int cmp(int i,int j){for(int k=1; k<=m; k++){if(a[i][k]<a[j][k]) return -1; else if(a[i][k]>a[j][k])return 1;}return 0;}

void swap(int i,int j){int\* tmp=a[i]; a[i]=a[j]; a[j]=tmp;}

void heapify(int i,int n){int L=2\*i; int R=2\*i+1; int maxIdx=i;

if(L<=n&&cmp(L,maxIdx)>0) maxIdx=L; if(R<=n&&cmp(R,maxIdx)>0) maxIdx=R;

if(maxIdx!=i){swap(i,maxIdx);heapify(maxIdx,n);}}

void buildHeap(){for(int i=n/2;i>=1;i--) heapify(i,n);}

void heapSort(){buildHeap();for(int i=n;i>=2;i--){swap(1,i);heapify(1,i-1);}}

void print(){for(int i=1;i<=n;i++){for(int j=1;j<=m;j++)printf("%d ",a[i][j]);printf("\n");}}

void freeMemory(){for(int i=1;i<=n;i++){free(a[i]);}}

int main(){input(); heapSort(); print(); freeMemory(); return 0;}

Sorting a list of records: #define N 100001 #define MAX\_LEN 11

typedef struct Candidate{int score;char code[MAX\_LEN];}Candidate; int n; Candidate\* a[N];

void input(){char code[MAX\_LEN]; n=0; while(1){scanf("%s",code); if(strcmp(code,"#")==0)break; int score; scanf("%d",&score); n++; a[n]=(Candidate\*)malloc(sizeof(Candidate)); a[n]->score=score; strcpy(a[n]->code,code);}}

void swap(int i,int j){Candidate\* t=a[i]; a[i]=a[j]; a[j]=t;}

void heapify(int i,int n){int L=2\*i; int R=2\*i+1; int minIdx=i; if(L<=n&&a[L]->score<a[minIdx]->score) minIdx=L; if(R<=n&&a[R]->score<a[minIdx]->score) minIdx=R; if(minIdx!=i){swap(i,minIdx); heapify(minIdx,n);}}

void buildHeap(){for(int i=n/2;i>=1;i--) heapify(i,n);}

void sort(){buildHeap();for(int i=n;i>=2;i--){swap(1,i); heapify(1,i-1);}}

void print(){for(int i=1;i<=n;i++){printf("%s %d\n",a[i]->code, a[i]->score);}}

void freeMemory(){for(int i=1;i<=n;i++){free(a[i]);}}

int main(){input(); sort(); print(); freeMemory(); return 0;}

Finding the greatest common subset of two sets: #define N 100001 int a[N], b[N]; int n,m;

void input(){scanf("%d%d",&n,&m); for(int i=1;i<=n;i++)scanf("%d",&a[i]); for(int j=1;j<=m;j++)scanf("%d",&b[j]);}

void swap(int\* arr,int i,int j){int t=arr[i]; arr[i]=arr[j]; arr[j]=t;}

void heapify(int\* arr,int i,int n){int L=2\*i; int R=2\*i+1; int maxIdx=i; if(L<=n&&arr[L]>arr[maxIdx]) maxIdx=L; if(R<=n&&arr[R]>arr[maxIdx]) maxIdx=R; if(maxIdx!=i){swap(arr,i,maxIdx); heapify(arr,maxIdx,n);}}

void buildHeap(int\* arr,int n){for(int i=n/2;i>=1;i--) heapify(arr,i,n);}

void sort(int\* arr,int n){buildHeap(arr,n);for(int i=n;i>=2;i--){swap(arr,1,i); heapify(arr,1,i-1);}}

int main(){input(); sort(a,n); sort(b,m); int cnt=0; int i=1; int j=1; while(i<=n&&j<=m){if(a[i]==b[j]){cnt+=1; i++; j++;} else if(a[i]>b[j]) j++; else i++;}printf("%d\n",cnt); return 0;}

Pairs of numbers with a given sum:

#define N 1000001 int a[N]; int n,Q;

void input(){scanf("%d%d",&n,&Q);for(int i=1;i<=n;i++){scanf("%d",&a[i]);}}

void swap(int i,int j){int t=a[i]; a[i]=a[j]; a[j]=t;}

void heapify(int i,int n){int L=2\*i;int R=2\*i+1; int maxIdx=i; if(L<=n&&a[maxIdx]<a[L]) maxIdx=L; if(R<=n&&a[maxIdx]<a[R]) maxIdx=R; if(maxIdx!=i){swap(i,maxIdx); heapify(maxIdx,n);}}

void buildHeap(){for(int i=n/2;i>=1;i--) heapify(i,n);}

void heapSort(){buildHeap();for(int i=n;i>=2;i--){swap(1,i); heapify(1,i-1);}}

int bSearch(int i,int j,int k){if(i>j)return -1;int m=i+(j-i)/2; if(a[m]==k)return m;

if(a[m]<k)return bSearch(m+1,j,k); else return bSearch(i,m-1,k);}

int main(){input(); heapSort(); int cnt=0; for(int i=1;i<=n-1;i++){int idx=bSearch(i+1,n,Q-a[i]); if(idx>-1) cnt=cnt+1;} printf("%d\n",cnt); return 0;}

Checking the existence: #define N 100001 int a[N];int n;

void input(){scanf("%d",&n);for(int i=1;i<=n;i++){scanf("%d",&a[i]);}}

void swap(int i,int j){int t=a[i]; a[i]=a[j]; a[j]=t;}

void heapify(int i,int n){int L=2\*i; int R=2\*i+1; int maxIdx=i; if(L<=n&&a[maxIdx]<a[L]) maxIdx=L; if(R<=n&&a[maxIdx]<a[R]) maxIdx=R; if(maxIdx!=i){swap(i,maxIdx); heapify(maxIdx,n);}}

void buildHeap(){for(int i=n/2;i>=1;i--) heapify(i,n);}

void heapSort(){buildHeap(); for(int i=n;i>=2;i--){swap(1,i); heapify(1,i-1);}}

int bSearch(int i,int j,int k){if(i>j)return -1; int m=i+(j-i)/2; if(a[m]==k)return m; if(a[m]<k)return bSearch(m+1,j,k);

else return bSearch(i,m-1,k);}

int main(){input();heapSort();char cmd[30];while(1){scanf("%s",cmd);if(strcmp(cmd,"#")==0)break;else if(strcmp(cmd,"check")==0){int k;scanf("%d",&k);int res=bSearch(1,n,k);if(res!=-1)printf("1\n");else printf("0\n");}}return 0;}