Solutions for Program Assignment 2

CSCI2100A

February 9, 2015

1 Exercise 2.16

1.1 Analysis

Easy simulation. Scan the final sequence from the beginning, then push all the carts smaller than the current cart into stack, then output "S" for the current cart. Keep scanning when you find a cart smaller than the current maximum number, then check whether it is on the top of the stack. If it is, pop, otherwise, output "impossible".

1.2 Sample code

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#define N 100000

typedef struct {
    int *data;
    int top;
    int size;
} Stack;

int createStack(Stack* astack, int size)
{
    astack->data=(int*)malloc(sizeof(int)*size);
    if (astack->data==NULL) return 0;
    astack->size = size;
    astack->top=-1;
    return 1;
}

void makeEmpty(Stack *astack)
{
    astack->top=-1;
}

int isEmpty(Stack* astack)
{
    if (astack->top<0) return 1;
    else return 0;
}
```

```
int isFull(Stack* astack)
{
    if (astack->top==astack->size-1) return 1;
    else return 0;
}

int top(Stack* astack)
{
    if (!isEmpty(astack)) return astack->data[astack->top];
    else return 0;
}

int pop(Stack* astack)
{
    if (!isEmpty(astack)) {
        int adata=top(astack);
        astack->top--;
        return adata;
    }
    else return 0;
}

int push(Stack* astack, int adata)
{
    if (!isFull(astack)) {
        astack->top++;
        astack->data[astack->top]=adata;
        return 1;
    }
    else return 0;
}

int main()
{
    Stack *track;
    char *ans;
    int T, n, cart, i, j, k, flag, p, max;

    track = (Stack*)malloc(sizeof(Stack));
    if (!createStack(track, N)) {
        printf("Out of Memory!\n");
        return 0;
    }
    ans = (char*)malloc(sizeof(char)*2*N);
    scanf("%d", &T);
    for (i=0;i<T;i++) {
        scanf("%d", &n);
        flag = 1;
        makeEmpty(track);
        p = 0;
        max = 0;
```

```
        for (j=0; j<n;j++) {
            scanf("%d", &cart);
            if (flag) {
                if (cart>max) {
                    for (k=max+1;k<cart;k++) {
                        push(track, k);
                        ans[p++]='I';
                    }
                    ans[p++]='S';
                    max = cart;
                }
                else if (cart<max) {
                    if (top(track)==cart) {
                        pop(track);
                        ans[p++]='O';
                    }
                    else flag = 0;
                }
                else flag=0;
            }
        }
        ans[p]='\0';  \*indicate the end of string, very important!!!*\
        if (flag) printf("%s\n", ans);
        else printf("Impossible\n");
    }
    free(track);
    free(ans);
    return 0;
}
```

To generate large random test cases, you can use the following incomplete code (you need to add headers and stack functions).

```
int main()
{
    Stack *track;
    int i, order, k;

    srand(time(NULL));
    track = (Stack*)malloc(sizeof(Stack));
    printf("%d", N);
    if (!createStack(track, N)) {
        printf("Out of Memory!\n");
        return 0;
    }
    k = 1;
    while (k<=N) {
        order = rand()%3;
        if (order == 0) { //straight through
            printf(" %d", k);
            k++;
        }
```

```
        else if (order == 1) { //push
            push(track, k);
            k++;
        }
        else {
            if (!isEmpty(track)) { //pop
                printf(" %d", top(track));
                pop(track);
            }
        }
    }
    while (!isEmpty(track)) {
        printf(" %d", top(track));
        pop(track);
    }
    printf("\n");
    return 0;
}
```

2 Exercise 2.18

2.1 Analysis

The algorithm is finding two smallest number to merge and repeat this process $n-1$ times. The difficulty is how you put the sum back to the array and keep the array sorted. A simple way is to insert it on the right position and shift all elements on the right by one. It costs $O(n)$ time. A better way is to use another array B to store all the sum. Since the next sum will be larger than the previous sum, you can put the sum in the end of B, which costs $O(1)$. Thus, the overall time complexity will be $O(n)$.

To save space, we combine B into the original array. Since the original array will be deleted from the beginning and B will be inserted from zero-length, we can just update the deleted numbers in original array to the sum.

2.2 Sample Code

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

int main()
{
        long long int a[1000000], ans, sum;
        int n, i, j, len_b, ia, ib;

        while (1) {
                scanf("%d", &n);
                if (!n) break;
                for (i = 0; i < n; i++) {
                        scanf("%lld", &a[i]);
                }
                if (n==1) {  \*special case*\
                        printf("0\n");
                }
```

```
                else {
                        ans = 0;
                        len_b = 0; \*current length of sum array*\
                        ia = 0; \*current position of smallest element in original array*\
                        ib = 0; \*current position of smallest element in sum array*\
                        for (i = 1; i < n; i++) { \*need to do n-1 merge in total*\
                                sum = 0;
                                for (j = 0; j < 2; j++) { \*find the smallest number twice*\
                                        if (ib == len_b) {
                                                sum += a[ia];
                                                ia += 1;
                                        }
                                        else if (ia == n) {
                                                sum += a[ib];
                                                ib += 1;
                                        }
                                        else {
                                                if (a[ib] < a[ia]) {
                                                        sum += a[ib];
                                                        ib += 1;
                                                }
                                                else {
                                                        sum += a[ia];
                                                        ia += 1;
                                                }
                                        }
                                }
                                a[len_b] = sum; \*to save space, we update in the original array*\
                                len_b += 1;
                                ans += sum;
                        }
                        printf("%lld\n", ans);
                }
        }
        return 0;
}
```