Aayush Shah
19BCE245
9 September 2021

Design and Analysis of Algorithms

Practical 4

• **Code :**

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

/*
     Returns maximum of two integers
*/
int max2(int a, int b){
     return (a>b) ? a : b;
}

/*
     Returns maximum of three integers
*/
int max3(int a, int b, int c){
     return max2(max2(a,b),c);
}

/*
     Prints subarray
*/
void printSubArray(int arr[], int startIndex, int endIndex){
     printf("\nSub Array : ");
     for(int i=startIndex;i<=endIndex;i++){
          printf("%d ",arr[i]);
     }
}
```

```
/* ------------------------------ */

/*
     Kadane's algorithm
     Solves the maximum subarray sum in O(N) time.
*/
int maxSubArraySum1(int arr[], int size)
{
     int maxTillHere = 0, maxBest = arr[0], counter = 0,
startIndex = 0, finalStartIndex = 0;
     int subArray[size];
     int maxCounter = 0;

     for(int i=0;i<size;i++){
          if(maxTillHere+arr[i]<arr[i]){
               maxTillHere = arr[i];
               counter = 0;
               startIndex = i;
          }
          else{
               maxTillHere += arr[i];
               counter++;
          }
          if(maxTillHere>maxBest){
               maxBest = maxTillHere;
               maxCounter = counter;
               finalStartIndex = startIndex;
          }
     }

     printSubArray(arr, finalStartIndex,
finalStartIndex+maxCounter-1);

//   printf("\nSubarray : ");
//   for(int i=finalStartIndex;i<finalStartIndex+counter;i++){
//        printf("%d ",arr[i]);
//   }

     return maxBest;
}

/* ------------------------------ */
```

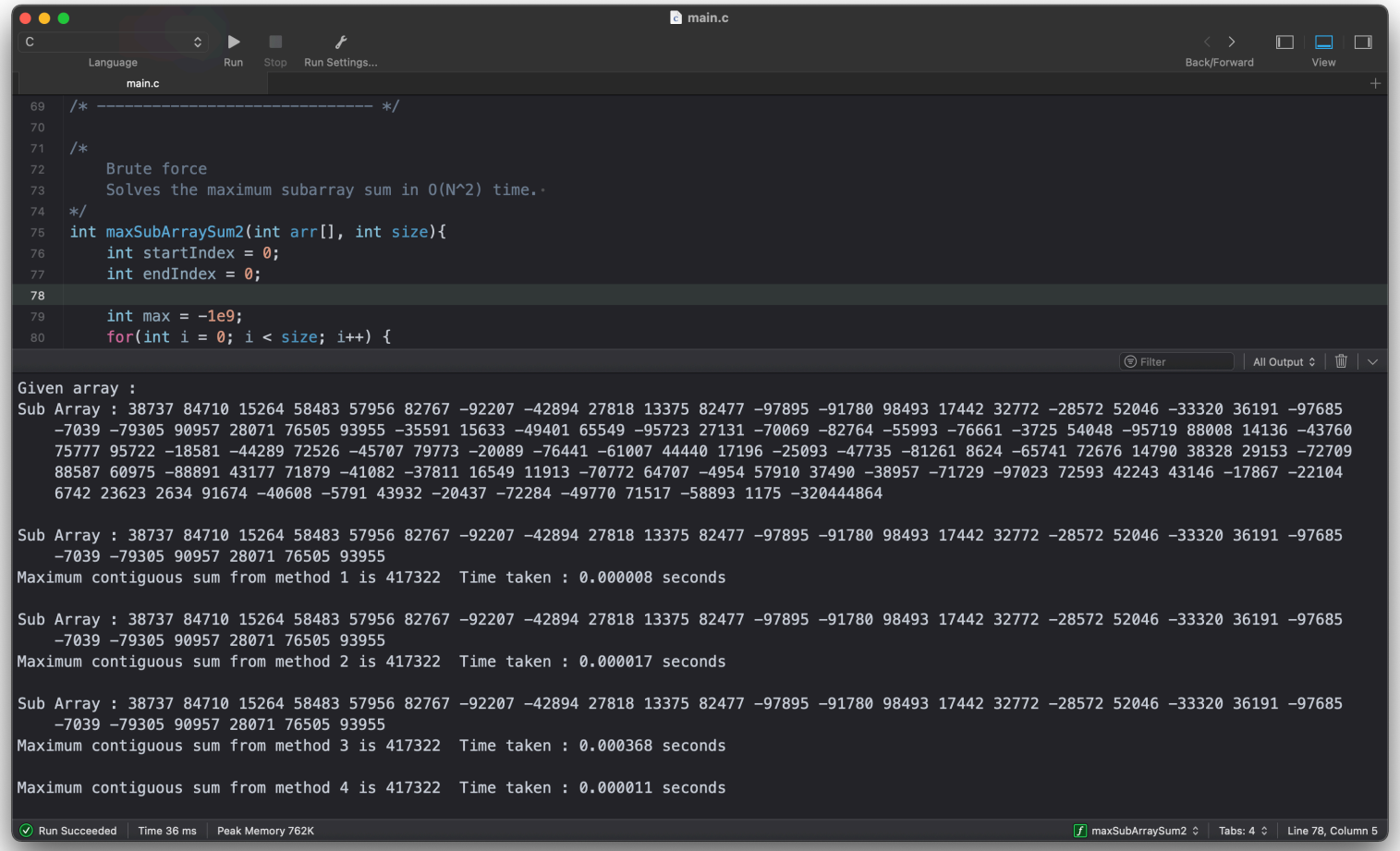
```
/*
     Brute force
     Solves the maximum subarray sum in O(N^2) time.
*/
int maxSubArraySum2(int arr[], int size){
     int startIndex = 0;
     int endIndex = 0;

     int max = -1e9;
     for(int i = 0; i < size; i++) {
          int sum = 0;
          for (int j = i; j < size; j++) {
               sum += arr[j];
               if (sum > max){
                    max = sum;
                    startIndex = i;
                    endIndex = j;
               }
          }
     }

     printSubArray(arr, startIndex, endIndex);

//   printf("\nSubarray : ");
//   for(int i=startIndex;i<=endIndex;i++){
//        printf("%d ",arr[i]);
//   }
     return max;
}

/* ------------------------------ */

/*
     Brute force (another approach)
     Sovles the maximum subarray sum in O(N^3) time.
*/
int maxSubArraySum3(int arr[], int size){
     int max = -1e9;
     int startIndex = 0, endIndex = 0;

     for (int i = 0; i < size; i++)
          for (int j = i; j < size; j++) {
               int sum = 0;
```

```
                for (int k = i; k <= j; k++)
                    sum += arr[k];
                if (sum > max){
                    max = sum;
                    startIndex = i;
                    endIndex = j;
                }
            }

    printSubArray(arr, startIndex, endIndex);

    return max;
}

/* ------------------------------ */

/*
    Divide and Conquer
    Solves the maximum subarray sum in O(N log(N)) time.
*/
int maxSumIncludingMid(int arr[], int l, int m, int h){
    int sum = 0;
    int left_sum = INT_MIN;
    for(int i=m; i>=l; i--){
        sum += arr[i];
        if (sum>left_sum)
            left_sum = sum;
    }

    sum = 0;
    int right_sum = INT_MIN;
    for(int i=m+1; i<=h; i++){
        sum += arr[i];
        if (sum>right_sum)
            right_sum = sum;
    }

    return max3(left_sum+right_sum, left_sum, right_sum);
}

int divideAndConquer(int arr[], int l, int h){
    if (l==h)
        return arr[l];
```

```
	int m = (l+h)/2;

	return max3(divideAndConquer(arr, l, m),
divideAndConquer(arr, m+1, h), maxSumIncludingMid(arr, l, m,
h));
}

int maxSubArraySum4(int arr[], int size){
	int l = 0;
	int h = size - 1;
	return divideAndConquer(arr, l, h);
}

/* ------------------------------ */

int main() {

	int lower = -100000;
	int upper = 100000;
	int count = 100;

	srand(time(0));

	int a[count];
	for (int i=0; i<count; i++) {
		a[i] = (rand() % (upper - lower + 1)) + lower;
	}

//	int a[] = {-5, 4,  6, -3, 4, -1};  //11
//	int a[] = {-2, -3,  4, -1, -2,  1,  5, -3};  //7
//	int a[] = {-2, -3, -4, -1, -2, -1, -5,  3};  //3
//	int a[] = {-2, -3, -4, -1, -2, -1, -5, -3};  //-1
//	int a[] = {-2, -3,  4, -1, -2, -1, -5, -3};  //4

	printf("Given array : ");
	printSubArray(a, 0, count);
	printf("\n");

	clock_t start, end;
	double cpu_time_used;
```

```
    int n = sizeof(a)/sizeof(a[0]);

    start = clock();
    printf("\nMaximum contiguous sum from method 1 is %d",
maxSubArraySum1(a, n));
    end = clock();
    cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
    printf("\tTime taken : %f seconds\n", cpu_time_used);
    start = clock();

    printf("\nMaximum contiguous sum from method 2 is %d",
maxSubArraySum2(a, n));
    end = clock();
    cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
    printf("\tTime taken : %f seconds\n", cpu_time_used);
    start = clock();

    printf("\nMaximum contiguous sum from method 3 is %d",
maxSubArraySum3(a, n));
    end = clock();
    cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
    printf("\tTime taken : %f seconds\n", cpu_time_used);
    start = clock();

    printf("\nMaximum contiguous sum from method 4 is %d",
maxSubArraySum4(a, n));
    end = clock();
    cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
    printf("\tTime taken : %f seconds\n", cpu_time_used);



    return 0;
}
```

- **Output :**


main.c
C
Language
Run
Stop
Run Settings...
Back/Forward
View
main.c
/* ------------------------------ */
/*
Brute force
Solves the maximum subarray sum in O(N^2) time.
*/
int maxSubArraySum2(int arr[], int size){
int startIndex = 0;
int endIndex = 0;
int max = -1e9;
for(int i = 0; i < size; i++) {
Filter
All Output
Given array :
Sub Array : 38737 84710 15264 58483 57956 82767 -92207 -42894 27818 13375 82477 -97895 -91780 98493 17442 32772 -28572 52046 -33320 36191 -97685
-7039 -79305 90957 28071 76505 93955 -35591 15633 -49401 65549 -95723 27131 -70069 -82764 -55993 -76661 -3725 54048 -95719 88008 14136 -43760
75777 95722 -18581 -44289 72526 -45707 79773 -20089 -76441 -61007 44440 17196 -25093 -47735 -81261 8624 -65741 72676 14790 38328 29153 -72709
88587 60975 -88891 43177 71879 -41082 -37811 16549 11913 -70772 64707 -4954 57910 37490 -38957 -71729 -97023 72593 42243 43146 -17867 -22104
6742 23623 2634 91674 -40608 -5791 43932 -20437 -72284 -49770 71517 -58893 1175 -320444864
Sub Array : 38737 84710 15264 58483 57956 82767 -92207 -42894 27818 13375 82477 -97895 -91780 98493 17442 32772 -28572 52046 -33320 36191 -97685
-7039 -79305 90957 28071 76505 93955
Maximum contiguous sum from method 1 is 417322 Time taken : 0.000008 seconds
Sub Array : 38737 84710 15264 58483 57956 82767 -92207 -42894 27818 13375 82477 -97895 -91780 98493 17442 32772 -28572 52046 -33320 36191 -97685
-7039 -79305 90957 28071 76505 93955
Maximum contiguous sum from method 2 is 417322 Time taken : 0.000017 seconds
Sub Array : 38737 84710 15264 58483 57956 82767 -92207 -42894 27818 13375 82477 -97895 -91780 98493 17442 32772 -28572 52046 -33320 36191 -97685
-7039 -79305 90957 28071 76505 93955
Maximum contiguous sum from method 3 is 417322 Time taken : 0.000368 seconds
Maximum contiguous sum from method 4 is 417322 Time taken : 0.000011 seconds
Run Succeeded
Time 36 ms
Peak Memory 762K
maxSubArraySum2
Tabs: 4
Line 78, Column 5