EXPLANATIONS OF CODES

```
read_all_file: takes a big arr and reads$stores all file to it

read_line: takes arr,bufferarr and reads a line from arr and stores it to bufferarr

new_max_array: takes curarr and maxarr and stores curarr to maxarr and updates maxsize

evaluate_line: takes bufferarr and curarr and while iterating bufferarr finds all
possibilites and stores&prints it through curarr and stores longest curarr to maxarr

clear_curarr: takes curarr and equalizes all its elements to zero

print_curarr: takes curarr and prints all its elements to console

clear_max_array: takes maxarr and equalizes all ites elements to zero

write to file: writes bufferarr+maxarr+maxsize to file
```

TEST CASES AND RESULTS

```
1 Array_inp: 2 6 5   | Array_outp: 2 6   | size: 2
2 Array_inp: 3 8 5 7 4 9   | Array_outp: 3 5 7 9   | size: 4
3 Array_inp: 4 5 2 7 6 8 9   | Array_outp: 4 5 7 8 9   | size: 5
4 Array_inp: 4 2 5 1 7   | Array_outp: 4 5 7   | size: 3
5 Array_inp: 8 1 5 6 2 7 9   | Array_outp: 1 5 6 7 9   | size: 5
6 Array_inp: 7 5 4 3 1 8   | Array_outp: 7 8   | size: 2
7 Array_inp: 1 5 8 2 3 4 9   | Array_outp: 1 2 3 4 9   | size: 5
```

TEST CASE 1: Input with that its last element is not the biggest one (Example 1)

EXPECTED RESULT: Since i control “j==arrsize” before going inside middle_loop there will be no problem.

RESULT: PASSED

TEST CASE 2: Input with that its last element is the biggest and first element is second biggest but inside array there is a longer possibility. (Example 5)

EXPECTED RESULT: Outer loop considers all possibilities for all elements of bufferarr as first element so there will be no problem.

RESULT: PASSED

TEST CASE 3: Since “2,6,5” and “2,6,5,4” in terms of this algorithm program should not print both cases.

EXPECTED RESULT: Since i used ‘continue’ at start of middle loop i don’t print same curarr’s to console.

RESULT: PASSED

COMMENTS

MISSING PARTS: My code only works with 1 digit numbers.

BONUS PARTS: I printed inner results to console.

OPTIMIZATION TRICKS: I controled “if(arr[j] < curarr[iter-1])” this way program skips 1 ‘for’ loop because that case could not be our (shortest to longest) case. My algorithm does not considers that ones as a possibility and saves 1 loop time.

You can use commas between numbers instead of spaces in the input file.

$T(n) = O(n^4)$

```c
int main(){
    int allarr[7][10] = {{2,6,5}, {3,8,5,7,4,9}, {4,5,2,7,6,8,9}, {4,2,5,1,7}, {8,1,5,6,2,7,9}, {7,5,4,3,1,8}, {1,5,8,2,3,4,9}};
    int l;
    for(l = 0;l < 7;l++){
        int i,j,k,iter = 0;
        int maxsize = 0;
        int cursize = 0;
        int arrsize = determineSize(allarr[l]);
        int arr[arrsize];
        memcpy(arr, allarr[l], arrsize * sizeof(int));
        int curarr[arrsize];
        int maxarr[arrsize];
        printf("Buffer: ");
        print(arr,arrsize);
        printf("\n");
        for(i = 0;i < arrsize;i++){
            iter = 0;
            curarr[iter++] = arr[i];
            for(j = i + 1;j < arrsize;j++){
                if(arr[j] < curarr[iter-1])
                    continue;
                for(k = j;k < arrsize;k++){
                    if(arr[k] > curarr[iter - 1])
                        curarr[iter++] = arr[k];
                }
                print(curarr,iter);
                printf("\n");
                if(maxsize < iter){
                    maxsize = iter;
                    int t;
                    for(t = 0;t < maxsize;t++)
                        maxarr[t] = curarr[t];
                }
                memset(curarr, 0, sizeof(curarr));
                iter = 0;
                curarr[iter++] = arr[i];
            }
            memset(curarr, 0, sizeof(curarr));
        }
        printf("Maxarr: ");
        print(maxarr,maxsize);
        printf("| maxsize = %d\n\n",maxsize);
        memset(maxarr, 0, sizeof(maxarr));
        memset(arr, 0, sizeof(arr));
    }
}
```


$O(1)$ => All spaces are constant

```
        .data
space: .asciiz " "          # 1 byte
endl: .asciiz  "\n"         # 1 byte
arr: .space 804             # 804 byte  read all lines from file
result: .asciiz "result.txt"  # 10byte  result filename
file: .asciiz "array.txt"     # 9byte   filename
input: .asciiz "Array_inp: "  # 11 byte
maximum_array: .asciiz " | Array_outp: "  # 15 byte
size: .asciiz " | size: "     # 9byte
bufferarr: .space 40          # 40 byte  buffer array for single line
curarr: .space 40             # 40 byte  current array to evaluate bufferarr and compare it with maxarr
maxarr: .space 40             # 40 byte  array with maximum length
iter: .word 0                 # 4 byte   to iterate through buffer array also through this we find 'curarr's size to compare it with maxsize
arrsize: .word 0              # 4 byte   size of array from read file
maxsize: .word 0              # 4 byte   size of array with maximum size
```

C AS PSEUDOCODE

```c
int main(){
    int allarr[7][10] = {{2,6,5}, {3,8,5,7,4,9}, {4,5,2,7,6,8,9}, {4,2,5,1,7}, {8,1,5,6,2,7,9}, {7,5,4,3,1,8}, {1,5,8,2,3,4,9}};
    int l;
    for(l = 0;l < 7;l++){
        int i,j,k,iter = 0;
        int maxsize = 0;
        int cursize = 0;
        int arrsize = determineSize(allarr[l]);
        int arr[arrsize];
        memcpy(arr, allarr[l], arrsize * sizeof(int));
        int curarr[arrsize];
        int maxarr[arrsize];
        printf("Buffer: ");
        print(arr,arrsize);
        printf("\n");
        for(i = 0;i < arrsize;i++){
            iter = 0;
            curarr[iter++] = arr[i];
            for(j = i + 1;j < arrsize;j++){
                if(arr[j] < curarr[iter-1])
                    continue;
                for(k = j;k < arrsize;k++){
                    if(arr[k] > curarr[iter - 1])
                        curarr[iter++] = arr[k];
                }
                print(curarr,iter);
                printf("\n");
                if(maxsize < iter){
                    maxsize = iter;
                    int t;
                    for(t = 0;t < maxsize;t++)
                        maxarr[t] = curarr[t];
                }
                memset(curarr, 0, sizeof(curarr));
                iter = 0;
                curarr[iter++] = arr[i];
            }
            memset(curarr, 0, sizeof(curarr));
        }
        printf("Maxarr: ");
        print(maxarr,maxsize);
        printf("| maxsize = %d\n\n",maxsize);
        memset(maxarr, 0, sizeof(maxarr));
        memset(arr, 0, sizeof(arr));
    }
}
```

think it as file read and store it to 'arr'

declaration of variables and we can think this as read 'buffer arr' from 'arr' in assembly

restart iter and fill arr[i] as first element

I explained this optimization and design part in report

fills currentarr if next element is bigger than latest

since it's filled print it to console

if current possibility is better than maxarr replace it

clear it for next time

restart iter and fill arr[i] as first element

clear it for next time

write max array to file

clear arrays for next outer loop