CMPT 125, Fall 2022

Midterm Exam - Solutions
October 24, 2022

Name_____

SFU ID: |__|__|__|__|__|__|__|__|__|

Problem 1	
Problem 2	
Problem 3	
Problem 4	
TOTAL	

Instructions:

1. Duration of the exam is 100 minutes.
2. Write your full name and SFU ID NUMBER **clearly**.
3. This is a closed book exam, no calculators, cell phones, or any other material.
4. The exam consists of four (4) problems. Each problem is worth 25 points.
5. Write your answers in the provided space.
6. There is an extra page at the end of the exam. You may use it if needed.
7. Explain all your answers.
8. ***Really, explain all your answers***.

Good luck!

Problem 1 [25 points]

a) [6 points] Will the program compile? If yes, what will be the output of the following program? If not, explain why.

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

void do_something(int* a, int n) {
     a = malloc(n*sizeof(int));
     for(int i=0; i<n; i++)
          a[i] = i*i;
}

int main() {
     int* ar;
     do_something(ar, 5);
     for(int i=0; i<5; i++)
          printf("a[%d] = %d\n", i, ar[i]);
     free(ar);
     return 0;
}
```

ANSWER: it will compile.
But do_something(ar, 5) will not assign to ar the array created in the function. The for loop will have unexpected behavior (might crash Also free(ar) will try to free memory that was not allocated, and might also crash.

b) [6 points] Will the code below compile? If yes, what will be the output? If not, explain why.

```
#include <stdio.h>

int main() {
    char s[] = {'a','b',0,'1','2','3','4',0,'x','y','z'};
    int ind=0;
    while (s[ind])
      ind = ind+1;
    printf("%s\n", s+ind+2);
    return 0;
}
```

ANSWER: it will compile.
After the while loop we'll have ind=2 (pointing to the first 0).
printf("%s\n", s+ind+2) will print "234".

For c) and d) consider the following function

```
#include <stdio.h>

long what(int n) {
  if (n==0)
    return 1;
  return what(n-1)+what(n-1);
}
```

c) [7 points] Explain in words the functionality of the function. Given examples of input-output pairs for this function, e.g., what will be the output on input 3,4,5?

ANSWER: Let's start with some examples:
what(0) = 1
what(1) = what(0)+what(0) = 2
what(2) = what(1)+what(2) = 4
what(3) = what(2)+what(2) = 8
what(4) = what(3)+what(3) = 16
It's not hard to see that what(n) = 2*what(n-1), and hence what(n) return 2^n.

d) [6 points] Rewrite the function without using recursion, so that it has the same functionality, and running time O(n).

```
long what(int n) {
  long res=1;
  for (int i=1; i<=n; i++ )
    res*=2;
  return res;
}
// the solution return pow(2,n) will also be accepted for full marks
```

Problem 2 [25 points]

a) [5 points] Consider the **Binary Search** algorithm. List all comparisons it makes on the input A = [2, 4, 6, 8, 9, 12, 14, 15, 18, 90, 99] when searching for 70. Explain your answer.

ANSWER:

1. We first compare 70 to 12 and go right to [14, 15, 18, 90, 99]
2. Then, we compare 70 to 18, and go right to [90, 99]
3. Then, we compare 70 to 90, and go right to []
4. Declare **NOT FOUND**

Step 3 could also compare to 99, and then go right to [90].
Then compare 70 to 90, and declare **NOT FOUND**.

b) [8 points] Show an array with the values {1,2,3,4,5,6,7,8} so that **Insertion Sort** makes:

- exactly 1 swaps in the first iteration of the outer loop,
- exactly 7 swaps in the last iteration of the outer loop,
- no swaps in other iterations.

(Since the array has 8 elements, the outer loop of Insertion Sort has 7 iterations)
Explain your answer.

ANSWER:
A bit of trial and error gives the array [3,2,4,5,6,7,8,1]
In the first iteration we swap 2 and 3 and get [2,3,4,5,6,7,8,1]
Then, there are no comparisons until the last iteration, when we handle 1
In the last iteration we push 1 all the way to the left, make 7 swaps and get [1,2,3,4,5,6,7,8]

c) [12 points] Implement a recursive version of Binary Search. The function gets an array A of length n and an element elt.

- If elt is in A, the function returns an index i such that A[i]==elt.
- If elt is not in A, the function returns -1.

You need to write a recursive implementation.
Explain your code if necessary.

```
int binary_search_rec(int* A, int n, int elt) {
// base cases
  if (n==0)
    return -1;
  if (n==1) {
    if (A[0]==elt)
      return 0;
    else
      return -1;
  }
// recursive calls
  int mid = n/2;
  if (elt==A[mid])
      return mid;
  else if (elt<A[mid]) // go left
      return binary_search_rec(A, mid, elt);
  else { // elt> A[mid] go right
    // going right is a bit tricky, because we need to "shift +mid"
    int res = binary_search_rec(A+mid , n-mid, elt);
          // note res is the index of elt in the subarray A[mid…n]
          // so we'll need to "shift it +mid" to obtain the
          // correct index in A
    if (res==-1) // if elt not found, return -1
      return -1;
    else
      return res+mid; // take the result and "shift it +mid"
  }









}
```

Problem 3 [25 points]

a) [20 points] *Write a function that gets an array of length n>0 of strings containing non-negative integers, and returns the index of the largest one. If there are two maximal elements, the function can return the index of any of them.*

```
typedef const char* const_str; // define a type for const string
int str_num_max(const_str const numbers[], int n);
```

For example,

`str_num_max(["9","0","1","0"],4)` *returns 0, because 9 is the largest.*
`str_num_max(["5","8","2","48","3","48","0","18"],8)` *returns 3 or 5.*
`str_num_max(["52","52","52","52","52","52"],6)` *returns any number 0…5.*
`str_num_max(["934"],1)` *returns 0.*
`str_num_max(["214378798","54190238674879806948","8"],3)` *returns 1.*
`str_num_max(["980715234549091284749273829","511","9","561"],4)` *returns 0.*

1. *You may assume that the input is always legal, i.e., the strings represent positive integers in the correct format (no unnecessary leading zeros), all numbers are distinct*, and n>0.*
2. *Note that the numbers may be larger than the maximum of `int` or `long`.*
3. *You may use standard library functions, e.g., functions from string.h.*
4. *You may write helper functions if needed. (If this hint is not clear, it means, you should definitely write helper functions to make your code look clean and readable)*

** This was a typo. The numbers may be equal, but if you assume the numbers are all distinct, I'll accept the solution*

```
// helper function that gets two strings containing integers
// if num1>num2, return>0
// if num1<num2, return<0
// if num1==num2, return=0
int str_cmpr_num(const_str num1, const_str num2);
// we'll postpone it for now, and use it to solve the problem.


// the solution using the helper function above
int str_num_max(const_str numbers[], int n) {
  int max_index = 0;
  for (int i=1; i<n; i++)
      if (str_cmpr_num(numbers[i], numbers[max_index])>0)
      max_index=i;
  return max_index;
}
```

```
// implementation of helper function that gets two strings
containing integers
// if num1>num2, return>0
// if num1<num2, return<0
// if num1==num2, return=0
int str_cmpr_num(const_str num1, const_str num2) {
  int len1 = strlen(num1);
  int len2 = strlen(num2);
  if (len1 != len2)
      return len1-len2;
  // if len1==len2 we compare digit by digit
  // starting from the most significant digit
  for (int i = 0; i < len1; i++)
      if (num1[i] != num2[i])
        return num1[i]-num2[i];
  // if reached here, the numbers are equal
  return 0;
}
```

b) [5 points] Use big-O notation to analyze the running time of your function. Explain your answer.

```
We make total n-1 comparisons, each comparison using str_cmpr_num.
The running time of each comparison is at most
O(length of the longest string).
Therefore the total running time is O(n * strlen(longest string)).
```

Problem 4 [25 points]

a) [15 points] Write a function that gets a string and removes from it all spaces (ascii code 32).

```
void remove_spaces(char *str);
```

For example, consider the following code.

```
char str[] = " ab   cd*1_2 @  ";
remove_spaces(str)
printf("%s", str);
```

It will print `"abcd*1_2@"`.

For full marks your function needs to run in time O(strlen(str)) and use O(1) extra space.

```
void remove_spaces(char *str) {
  // idea: we'll have two indices:
  // read - this will be the index of the next char we read
  // write - this will be the index of the next char we write
  int read = 0;
  int write = 0;
  while (str[read] != '\0') {
    if (str[read] != ' ') {
      str[write] = str[read] ;
      write++;
    }
    read++;
  }
  str[write] = '\0';
}

// the body of the loop could also be written as
    if (str[read] != ' ')
      str[write++] = str[read] ;
    read++;
```

b) [7 points] Write a function that gets a string and checks if it is a palindrome of odd length.

```
bool is_odd_palindrome(const char* str);
```

For example:

- *On input* `"a2z@d@z2a"` *the function returns* `true`*.*
- *On input* `"ae11&edwdccek"` *the function returns* `false`*.*
- *On input* `"abccba"` *the function returns* `false`*, since the length is not* ***odd****.*

```
// the idea is pretty straightforward:
// if length is even, return false
// otherwise compare str[i] with str[n-1-i] for all i

bool is_odd_palindrome(const char* str) {

  int = strlen(str);

  // if length is even, return false
  if (n%2==0)
     return false;
  for (int i = 0; i < (n-1)/2; i++)
     if (str[i] != str[n-1-i])
       return false;

  return true;
}
```

[3 points] What is the running time of your function? Use big-O notation to state your answer. Give the tightest possible answer. Explain your answer.

ANSWER:
The running time is O(n), where n=strlen(str). This is because

- The running time of strlen(str) is O(n)
- The loop reads each char in str exactly once - total O(n)

Extra page

Empty page