CS 240 Data Structure

Spring 2018 Exam III

04/25/2018

This exam contains **BST, Stack, Quicksort and an arbitrary sorting algorithm.** That is 4 sections.

A) Stack
   a. Trace the code to predict the output of the code
   b. Some default function call, let say pop_push, is not available; please use other functions calls to implement the missing function
   c. If the program won't compile, simply say so.

B) BST in an array
   a. Trace the code to predict the output of the code
   b. Interpretation of a specific line of code
   c. Complexity of the algorithm

C) Quicksort in an array
   a. Trace the code to predict the output of the code
   b. Swap code and its effect

D) Another sorting algorithm
   a. Trace the code to predict the output of the code
   b. Mistakes in duplication
   c. Fix the mistake in duplication

⇨ Good luck and do your best.

⇨ When you are lost in pointers and addresses, the best way to attack that is simply drawing the diagram with respect to address of the variable that the pointer points to, and the pointer's its own address and the dereferencing operator *.

Name:

Score: /80

A) Stack (see the implementation in Appendix A) : 3*4 + 8pts

(1) Please write out the output 1 from that main function:

(2) Please write out the output 2 from that main function:

(3) Please write out the output 3 from that main function:

(4) When you use vector, that good thing about it is the dynamic growth. Let say, now you are not allowed to use the pop_back function of vector but let say you can use some removeLoc function of vector. Example, vector a={1,2,3,4}; a.removeLoc(3) will get ride of 4 and the new vector a={1,2,3}. And you also have the function size, that is if b={2,1,4}, then b.size() = 3; How would you do to implement the pop_back by use of removeLoc() and size()?

```
template <class T>
void Stack<T>::pop ()
{  // write your implementation here
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
}
```

B) Binary Search Tree (see the implementation in Appendix B): 3*6+ 8

(1) Please print out the parameters passed of each recursion call when it is on the search for value 3; For instance, the first set of

	first	Last	Arr[med]
Initial	0	10	31

2nd

3rd

(2) there is a line of code if (first > last) return -1; what does it mean logically?

(3) What is the complexity in finding an element in such a sorted array provided that array size is N

(C) Quick sort(see the implementation in Appendix C)(10 + 10 points)

(1) For the line cout<< "the value of i is " << i << "and the value of j is " <<j<<endl; please predict the first five results printed out from this line of code for i and j .

1st occurrence: the value of i is and the value of j is

2nd currence: the value of i is and the value of j is

3rd occurrence: the value of i is and the value of j is

4th occurrence: the value of i is and the value of j is

5th occurrence: the value of i is and the value of j is

(2) for those two lines (a) if (left < j) quickSort(arr, left, j); (b) if (i < right) quickSort(arr, i, right);

Will it affect the result if we swap the order, i.e. instead of (a)(b), I do (b)(a)? Why?

(D) Big O notation (see the implementation in Appendix D): 5 + 2 + 3 + 10 points

In Appendix D, as we saw before, it works fine because there are no duplicated values in the array. Let say now we have a new array `int test[] = {10, 5,20, 20, 5, 6, 200};`

1. Please predict the output array

2. What is the problem in the output due to duplicated values? Was the result array having empty spots base on your answer in (1)?

3. What can you do in the code to avoid this duplicated problem? (please directly write on appendix D source code page, not here)

Appendix A

(assume all packages are included)

```
using namespace std;
template <class T>
class Stack {
  private:
    vector<T> elems;
  public:
    void push(T elem);
    void pop();       T top();
   bool empty() { return elems.empty(); }   };
template <class T>
void Stack<T>::push (T elem) {    elems.push_back(elem);   }

template <class T>
void Stack<T>::pop () {    if (elems.empty()) {  throw out_of_range("Stack<>::pop(): empty stack"); }
elems.pop_back();  }

template <class T>
T Stack<T>::top (){  if (elems.empty()) { throw out_of_range("Stack<>::top(): empty stack"); }  return elems.back();
}
int main() {
    Stack<int>       intStack;    Stack<string> stringStack;    intStack.push(7); intStack.push(8); intStack.push(12);
cout << intStack.top() <<endl;      cout << intStack.top() <<endl;      // expected output 1
While(!intStack.empty()){cout << intStack.pop() <<endl;}       // expected output 2
stringStack.push("hello");  stringStack.push("world");    stringStack.push("CS 240 Spr 2018");
cout << stringStack.top() << std::endl; cout << stringStack.pop() << std::endl; cout << stringStack.pop() <<
std::endl; //expected output 3
return 0; }
```

Appendix B: BST using array

```
(assum all packages are included)
int binarysearch(int ary[], int first, int last, int target);
int main()
{   int findit;     int myarray[] = {2,3,4,5,21, 31,44, 55, 67,89,451};
  cout<<"Please enter the element you wan to search:"<<endl; // Let say I input 3
    cin>> findit;
    int k = binarysearch(myarray, 0, 10, findit);
       if(k==-1)   cout<< "cannot find";
       else     cout << "found at locaiton " << k <<endl;
    return 0;
}
int binarysearch(int ary[], int first, int last, int target)
{  int med = (first + last)/2; // expect 2
    if (first > last)    return -1;
    if(ary[med] == target)       return med;
     if (ary[med] > target)     binarysearch(ary, 0, med-1, target);
    if (ary[med] < target)       binarysearch(ary, med+1, last, target);
}
```

Appendix C: Quick Sort

```
(assume all packages are all included)
void quickSort(int arr[], int left, int right) {
    int i = left, j = right;
    int tmp;
    int pivot = arr[(left + right) / 2];
    while (i <= j) {
        while (arr[i] < pivot)              i++;
        while (arr[j] > pivot)              j--;
       cout<< “the value of i is “ << i << “and the value of j is “ <<j<<endl;
        if (i <= j) {
            tmp = arr[i];              arr[i] = arr[j];              arr[j] = tmp;
            i++; j--;
}    };
    if (left < j)          quickSort(arr, left, j);
    if (i < right)          quickSort(arr, i, right);
}

int main(void)
{    int x[] = {2,2,3,45, 3, 6, 4, 7};
    int thesize = sizeof(x)/4;
    quickSort(x, 0,thesize-1);
    return 0;
}
```

Appendix D:

```
int main()
{   int test[] = {10, 15, 20, 5, 6, 200};    int thesize = sizeof(test)/4;    int result[thesize];
   int HowManySmallerThanMe = 0;

  for(int i = 0; i < thesize; i++)
   {    HowManySmallerThanMe = 0;
          for(int j=0; j<thesize; j++)
             {  if(test[j] < test[i])
                    HowManySmallerThanMe++;    }
      result[HowManySmallerThanMe] = test[i];
   }
   return 0;
}
```