EECE 7205 HW2

Yaqiao Liu
NUID 001279512

Q1.

a.

BINARY_SEARCH(A,X)	Best case	Worst case
left = 0	1	1
right = A.length - 1	1	1
while left <= right	1	$(\log_2 n+1)+1$
n = right – left +1	1	$\log_2 n+1$
mid = left + n/2	1	$\log_2 n+1$
If A[mid] = x	1	$\log_2 n+1$
Return mid	1	
If A[mid] < x		$\log_2 n+1$
left = mid + 1		$\log_2 n+1$
else		
right = mid -1		
return -1		1
Total temporal cost	7	$6\log_2 n+10$

b.

Proof:

If $\exists$ c_1, c_2, and n_0 | $c_1 * \log_2 n <= f(n) <= c_2 * \log_2 n$ $\forall$ $n >= n_0$, then we can say the asymptotic cost of an iterative binary search in the worst case is $T = \theta(\log_2 n)$

So here we can get

$c_1 * \log_2 n <= 6\log_2 n+10 <= c_2 * \log_2 n$

Then

$c_1 * \log_2 n <= 6\log_2 n+10$ we can choose $n_0 = 4$, $c_1 = 10$

It proves the lower bound is $\log_2 n$ ($\Omega(\log2n)$)

$c_2 * \log_2 n >= 6\log_2 n+10$ we can choose $n_0 = 4$, $c_2 = 12$

It proves the upper bound is $\log_2 n$ (O (log2 n))

There exists a set of c_1, c_2, and n_0 meeting the inequation.

To sum up, we can say the asymptotic cost of an iterative binary search in the worst case is

$T = \theta(\log_2 n)$

Q2.

b) Class constructors and destructors:

constructor is a function with the same name as the class. It has no return value, and it is automatically called when an objected is created. The declaration of constructor is in the header file(shape.h), and the implement of the constructor is in the source file(shape.cc). It has an argument which is a private field “name” of the class shape. The name of object would be initialized during calling of constructors.

Destructor also has the same name as the class. Here we omit the destructor function, but it would be automatically invoked before the object is destroyed.

c) Class instantiation

In this project, we use the dynamic class instantiation. We use the new keyword in the main file.(main.cc) An object is an instantiation of a class, and it could be a variable, which has all the fields and function from the specific class.

d) Virtual function:
A virtual member is a member function that can be redefined in a derived class, while preserving its calling properties through references. For example, we use the virtual function in the class shape, we have a function named Print(), here we use the keywords virtual. It means that we can override it in a child class, for example, in derived class(class circle in Circle.h), we have the same function Circle::Print. In a word, a virtual function makes its class a polymorphic base class.

e) Pure virtual function:
Pure virtual function is a virtual function whose declaration ends in =0; sometimes we don’t know the implements of the function, so it cannot be provided in a base class. A pure virtual function could make a class abstract, in other words, abstract class cannot be instantiated. In this project, we use the pure virtual function in shape.h, and we use a virtual float getArea() = 0, because we cannot know the specific area calculation method of different shapes at this time.

f) Abstract class
Abstract classes can only be used as base classes, and thus are allowed to have virtual member functions without definition (known as pure virtual functions). The syntax is to replace their definition by =0 (an equal sign and a zero). Here the class Shape is an abstract class(in shape.h), and it cannot be instantiated. And the child class must override the virtual function(getArea()).

g) Polymorphism
Polymorphism allows a pointer to a derived nverts into a parent class. For example, in the main.cc, we define a pointer to circle object *c and a pointer to shape object *s. Here we can covert the c to s, because the class circle is a child class extending class shape. We called this feature as Polymorphism.

h) Include guards.

in every header file, we all use the include guards. For example, in shape.h, we use

```
#ifndef SHAPE_H
#define SHAPE_H
#endif
```

In this way, we can avoid duplicating the same include-guard macro name in different header files.

Q3.

When I used the keyword virtual, the output could be

```
Circle with radius 10
the circle has been destroyed
```

it shows that the virtual destructor has been invoked, and the object from derived class has been destroyed. The memory has been released.

But when I didn't use the keyword virtual, the output is:

```
Circle with radius 10
```

which means that the derived class destructor has not been called. In other words, the memory for child class object is not released. It can cause the memory leak, which is very dangerous.

The reason of difference is due to we use the dynamic memory allocation. C++ specified that when a derived class object is deleted through a pointer to a base class with a non-virtual destructor, results are undefined. What happens at runtime is that the derived object would not be destroyed.

The first version (with virtual) is better, because it can prevent memory leak.

The first case:
1. allocate the memory
2. call to constructor Shape::Shape();
3. call to constructor Circle::Circle();
4. call to destructor Circle::~Circle();
5. call to destructor Shape::virtual ~Shape();
6. Free the memory

The second case:
1. allocate the memory
2. call to constructor Shape::Shape();
3. call to constructor Circle::Circle();
4. call to destructor Shape::~Shape();

5. Free the memory