work will receive 0 credit.

Part (a)

```
void f1(int n)
{
  int i=2;
  while(i < n){
      /* do something that takes O(1) time */
      i = i*i;
  }
}
```

Part (b)

```
void f2(int n)
{
  for(int i=1; i <= n; i++){
    if( (i % (int)sqrt(n)) == 0){
      for(int k=0; k < pow(i,3); k++){
        /* do something that takes O(1) time */
      }
    }
  }
}
```

Part (c)

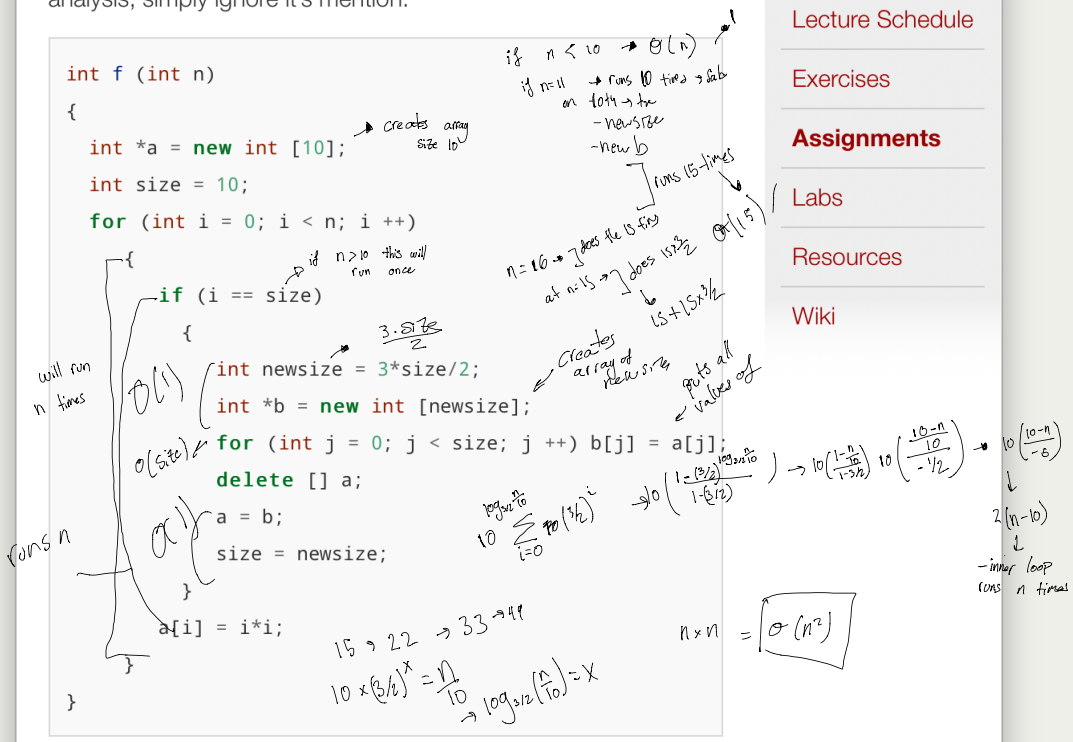
```
for(int i=1; i <= n; i++){
  for(int k=1; k <= n; k++){
    if( A[k] == i){
      for(int m=1; m <= n; m=m+m){
        // do something that takes O(1) time
        // Assume the contents of the A[] array are not char
      }
    }
  }
}
```

Part (d)

```
    }
  }
}
```

Part (d)

Notice that this code is very similar to what will happen if you keep inserting into an ArrayList (e.g. `vector`). Notice that this is **NOT** an example of amortized analysis because you are only analyzing *1* call to the function `f()`. If you have discussed amortized analysis, realize that does NOT apply here since amortized analysis applies to *multiple* calls to a function. But you may use similar ideas/approaches as amortized analysis to analyze this runtime. If you have NOT discussed amortized analysis, simply ignore it's mention.



```
int f (int n)
{
  int *a = new int [10];
  int size = 10;
  for (int i = 0; i < n; i ++)
  {
    if (i == size)
    {
      int newsize = 3*size/2;
      int *b = new int [newsize];
      for (int j = 0; j < size; j ++) b[j] = a[j];
      delete [] a;
      a = b;
      size = newsize;
    }
    a[i] = i*i;
  }
}
```

Problem 4 - Linked List Recursion Tracing (13%)

Consider the following C++ code. For the following questions, all of the points for this problem will be assigned based on your explanation, since we have full faith in your ability to run this program and copy down the answer.

To show work, you can draw a call tree or box diagram of the