Part a)

```
void f1(int n)
{
    int i = 2;
    while (i < n) {
        i = i * i;
    }
}
```

n = 2
runtime = 1

n = 4
runtime = 2

n = 5
runtime = 2

runtime = $O(\log_2 n)$

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++) {
            }
        }
    }
}
```

⇒ If n = 1, we enter the inner for loop every time we iterate through the outer for loop. → n times

runtime = $O(n)$

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) {
      }
    }
  }
}
```

→ $\log_2 n$

→ we can enter the inner for loop at most n times since k can be at most k so the if statement can be the for at most n times.

inner for loop → $\log_2 n$ → can be entered at most n times → $O(n \log_2 n)$

Part d)

```
int f (int n) {
  int *a = new int [10]; T(10)
  int size = 10;
  for( int i = 0; i<n ; i++) {
    if (i == size) {                                15
      int newsize = 3 * size/2;
      int b* = new int[newsize];
      for (int j = 0; j<size; j++) b[j]=a[j];
      delete []; T(10)          10
      a = b; T(15)
      size = newsize;
    }
    a[i] = i*i;
  }
}
```

All of the functions related to size take constant time as size is set as 10 in the beginning. The outer forloop is what depends on the input so the runtime becomes O(n).