1) def max difference (array):

n = len(array)

sorted_array = array.sort()

sublist_one = []
sublist_two = []

sum_one = 0
sum_two = 0

for i in range (n//2):

sublist_one.append (sorted_array[i])

sum_one += sorted_array[i]

for i in range (n//2, n):

sublist_two.append (sorted_array[i])

sum_two += sorted_array[i]

return sublist_one, sublist_two, absolute value (sum_one - sum_two)

Time complexity is O(n log(n))

1.2.

```
def difference (array):

    n = len(array)

    sorted_array = array.sort()

    sublist_one = []
    sublist_two = []

    sum_one, sum_two = 0, 0

    for i in range(n):

        if sum_one <= sum_two:

            sublist_one.append(sorted_array[i])
            sum_one += sorted_array[i]

        else:

            sublist_two.append(sorted_array[i])
            sum_two += sorted_array[i]
```

~~return sublist_one, sublist_two, absolute_value(sum_one - sum_two)~~

```
    return (sublist_one, sublist_two, absolute_value(sum_one - sum_two))
```

Time complexity is $O(n \times \log(n))$

2.

1) Best case:

when the first loop only loops once

So $i == n$ in the first run.

So Best case = $O(1)$.

2). Worst case will be it will run loops run from 1 to n.

worst case time complexity = $O(n^4)$.

5.

```
int any_equal (int n, int A[][]) {

unordered_set <int> seen;

  for (int i=0; i<n; i++) {
      for (int j=0; j<n; j++) {
          int element = A[i][j]
          if (seen.count(element) > 0)
          {
            return 1.
          }
          seen.insert(element);
      }
  }
  return 0;
}
```

3.

Bubble sort:

let Black ball = "1".
let White ball = "0".

given an Array of A= 2n.

loop through from 2n-1 to one in reverse order.

Compare the index(i) from the first possem (0)

if j find a element ("0"), that is smaller than the previous element (1), the loop will run and the elements will be swapped.

This will run until it makes sure all the "0"s are before the "1"s.

The time complexity will be $O(n^2)$.

The worst case count of the algorithm is $N\left(\frac{N}{2}+1\right)$. The average of the list should be between $\frac{N}{2}$ and $\left(\frac{N}{2}+1\right)$ so if the if condition is true, it will be summed $N+\left(\frac{N}{2}+1\right)$ times which is the worst case.

4.

line 2 from 1 to N

the best case will be condition is $(\frac{N}{2}+1)/2$. This would make the if statement true since there should be $\frac{N}{2}+1$ elements that are greater than the average.

the worst case count of the algorithm is $N(\frac{N}{2}+1)$. the average of the list should be between $\frac{N}{2}$ and $(\frac{N}{2}+1)$ so if the if condition, if count+1 > $\frac{N}{2}$ it will be summed $N+(\frac{N}{2}+1)$ times which is the worst case.

5.

a.

first loop $i < n^2 + 1$

second loop $j < n^{3/5}$

Total cost = $\frac{n^5}{5}$

b. while $(n^2 > 0)$

$j = j - 2$.

Total cost : $\frac{n^2}{2}$

c. while $(j \leq n^2)$

$j = \sum x j$.

$n = 2^k$.

cost : $\log_2(n)$

6.

a. $4^{2n+1} \in \Theta(16^n)$.

$$\lim_{n\to\infty} \frac{4^{2n+1}}{16^n} \to \lim_{n\to\infty} \frac{4^{2n} * 4}{16^n} \to \lim_{n\to\infty} \frac{(2^2)^{2n} \cdot 4}{(4^2)^n}$$

$$\to \lim_{n\to\infty} \frac{2^{4n} \cdot 4}{4^{2n}} \to \lim_{n\to\infty} 2^{4n-2n} = \lim_{n\to\infty} 2^{2n} \to \infty$$

4^{2n+1} is upper bound of 16^n.

$$\lim_{n\to\infty} \frac{4^{2n+1}}{16^n} \to \lim_{n\to\infty} \frac{(2)^{4n} \cdot 4}{4^{2n}} \to \lim_{n\to\infty} 2^{4-2n} \lim_{n\to\infty} 2^{2n}$$

$\to \infty$

This shows that 4^{2n+1} is $O(16^n)$.

b.

$$n^5 - n^2 + 2n \geq c \cdot n$$

$$n^5 - n^2 + 2n \geq n^5$$

$$\lim_{n\to\infty} \frac{n^5 - n^2 + 2n}{n} \geq c$$

$$\lim_{n\to\infty} \frac{n^5 - n^2 + 2n}{n} \to \lim_{n\to\infty} \frac{n^5}{n} - \frac{n^2}{n} + \frac{2n}{n}$$

$$\lim_{n\to\infty} \frac{n^5}{n} \to \lim_{n\to\infty} n^4 \to \infty$$

$$\lim_{n\to\infty} \frac{n^2}{n} \to \lim_{n\to\infty} n \to \infty$$

$$\lim_{n\to\infty} \frac{2n}{n} \to \lim_{n\to\infty} 2 \to 2.$$

$$\infty - \infty + 2 = \infty.$$

So it is bounded by $\Omega(n)$.

7.

$10n + 8 \in O(n^3)$

$10n + 8 \notin \Omega(n^3)$

let $f(n) = 10n + 8$

so $10n + 8 \leq c \times g(n)$

$10n + 8 \leq 10n^2$

Thus, $10n + 8 \in O(n^2)$ $g(n) = n^2$

let $10n + 8 = f(n)$

So $10n + 8 \geq 10 + 8$

This is true for $c = 18$ $g(n) = 1$

$g(n) \in \Omega(n^2)$.

8.

$f+g = O(f)$ $\quad g = O(f)$

$f+g = o(f)$

by definition of $g = o(f)$, there exist positive constant c and n_0 such that $n \geq n_0$, $g(n) \leq c \cdot f(n)$.

We need to show that $f+g$ is $o(f)$, which means we need to find c and N such that $n \geq N$,

$(f+g)(n) \leq c \cdot f(n)$.

So $(f+g)n = f(n) + g(n)$

i.e $f(n) + g(n) \leq f(n) + c * f(n)$

$f(n) + g(n) \leq (1+c) \cdot f(n)$.

Therefore $f+g = o(f)$.

$f+g = \Omega(f)$:

To show that $f+g$ is $\Omega(f)$, we need to find constant C' and N' such that for all $n \geq N'$, $(f+g)(n) \geq c' \cdot f(n)$.

$(f+g)(n) = f(n) + g(n)$.

since $g = o(f)$ then $(f+g)(n) = f(n) + g(n) \geq f(n) + c * f(n)$.

let $(f+g)(n)$ for $n \geq n_0$:

$(f+g)(n) = f(n) + g(n) \geq f(n) + c' f(n)$

$(f+g)(n) \geq (1+c) \cdot f(n)$.

$(f+g)(n) \geq c'$. So $f+g = \Omega(f)$.

9.

loop invariant:

ith iteration from i=1 to i=n-1

the array A[0:i] has the square roots of the cumulative sum of the squares of the first i elements of the array X.

Before i = 1

we set A[0] to X[0] and S to X[0] x X[0]

After each loop: i = (1 <= i < n-1)

A[0:i] contains the square roots of the cumulative sum of the squares of the first i elements of X.

After the loop ends

i = n

A[0:n-1] contains the square roots of the cumulative sum of the squares of the first n-1 elements of X.

we can conclude the above is correct since the loop invariant holds true for all three steps. in terms of [0,n-1]

10.

1) time complexity is $n \times n \times n = O(n^3)$.

[redacted]

2)

```
def calculate matrix (A):
    n = len(A)
    B = [[0] * n for _ in range(n)]        n
    # create matrix
    B[0][0] = A[0]
    for j in range(1,n):                   n
        B[0][j] = B[0][j-1] + A[j]

    for i in range(1,n):
        for j in range(i,n):
            B[i][j] = B[i-1][j] + A[i-1]
return B
```

Time complexity $O(n^2)$.

11.

1). Each minute the president gives student with "A" a opportunity to come to the podium. After each young person goes up to the podium, students with "A" can deduce that their own label by observing around. Eventually all 5 students with A will come to the podiums.

lets solve this by induction:

Base case:
minute 1 : no students come to the podium with "A"
since they are not certain about their label.

Inductive hypothesis:
Assume after k minutes, where k is the positive integer less than or equal to 5 all students with A figured out their label and went to the podium.

inductive step:
By minute (k+1), if no students with label "A" comes to the podium, then students with label "A" will realize they are among those of "A" and will come to the podium.

Here do we are done by solving this with induction.