```python
li = [12, 11, 13, 5, 6]

def binary_search(search_space, target):

    s = 0
    e = len(search_space) - 1

    # We will run loop while s <= end
    while s <= e:
        # find mid

        mid = (s + e)//2
        # compare
        if target == search_space[mid]:
            return mid
        elif target < search_space[mid]:
            # discard right
            e = mid - 1
        else:
            # discard left
            s = mid + 1
    return "Not found"



# Quiz

n=128
a = 0
i = n
count = 0
while (i > 1):
    count += 1
    a += i
    i //= 2
print(count)
```

```
7
```

```python
for i in range(1, 1000):
    print(i)
    break
```

```
1
```

Prime number

```python
# Give true if a number is prime else False

def prime_check(n):
    for i in range(2, n):
        if n % i == 0:   # This will be true only if n has a divisor
            return False
    return True

prime_check(21)

False

prime_check(11)

True

prime_check(99)

False

# Time complexity: O(n)



# Improvised version

def prime_check(n):
    for i in range(2, n//2 + 1):
        if n % i == 0:   # This will be true only if n has a divisor
            return False
    return True

prime_check(19)

True

# n -> n/2

# Time complexity: O(n)

prime_check(73)

True



# Checking till square root of n

def improvised_prime(n):

    for i in range(2, int((n ** 0.5)) + 1):
        if n % i == 0:
```

```python
            return False
    return True
```

```python
improvised_prime(99)
```

False

```python
improvised_prime(143)
```

False

```python
improvised_prime(73)
```

True

```python
# running almost sqrt(n) times
```

```python
# Time complexity: O(sqrt(n))
```

```python
# Which is better: log(n) or sqrt(n)
```

```python
# Prime : A number having only 1 and itself as facotrs
```

```python
n = 10

for i in range(1, n):
    if n % i == 0:
        print(i)
```

1
2
5

```python
# Finding factors between 2 to n - 1
```

```python
n = 10

for i in range(2, n):
    if n % i == 0:
        print(i)
```

2
5

```python
n = 11

for i in range(2, n):
    if n % i == 0:
        print(i)
```

Euclidean Distance

```
# (2, 5), (5, 10)

d = ((5 - 2) ** 2 + (10 - 5) ** 2) ** 0.5

d

5.830951894845301



Pa = (2, 5)
Pb = (5, 10)

x1, y1 = Pa
x2, y2 = Pb

print(x1, y1)

2 5

def euclidean_distance(Pa, Pb):

    x1, y1 = Pa
    x2, y2 = Pb

    d = (x2 - x1) ** 2 + (y2 - y1) ** 2
    d = d ** 0.5

    return d

euclidean_distance(Pa, Pb)

5.830951894845301



# use distance function to find the distances

# Find the distances of all

destinations = [(2, 5), (8, 3), (-1, 5), (0, 8), (6, 6)]
location = (1, 1)

for i in destinations:
    print(i)
```

```
(2, 5)
(8, 3)
(-1, 5)
(0, 8)
(6, 6)
```

```
for i in destinations:
    print(i, location)
```

```
(2, 5) (1, 1)
(8, 3) (1, 1)
(-1, 5) (1, 1)
(0, 8) (1, 1)
(6, 6) (1, 1)
```

KNN

```
# Finding distances from location to destinations

for i in destinations:
    print(euclidean_distance(i, location))
```

```
4.123105625617661
7.280109889280518
4.47213595499958
7.0710678118654755
7.0710678118654755
```

```
for i in destinations:
    d = euclidean_distance(i, location)
    print(d)
```

```
4.123105625617661
7.280109889280518
4.47213595499958
7.0710678118654755
7.0710678118654755
```

```
distances = []
for i in destinations:
    d = euclidean_distance(i, location)
    distances.append(d)

print(min(distances))
```

```
4.123105625617661
```

```python
# K insertion sort iterations

arr = [12, 11, 13, 5, 6]
k = 2

def insertion_sort(a, k):
    n = len(a)

    # Lets run for k iterations
    for i in range(1, k+1):
        index_to_insert = i
        j = i - 1

        while j >= 0:
            # Move left and compare values
            if a[j] < a[index_to_insert]:
                break
            # In case we need swap, update values
            a[j], a[index_to_insert] = a[index_to_insert], a[j]
            index_to_insert = j
            j -= 1
    return a[n//2]

insertion_sort(arr, k)
```

```
13
```

```python
# lexicographical selection sort

# a
# b

"a" < "b"
```

```
True
```

```python
"app" < "axe"
```

```
True
```

```
# abode
# app
# apple
# axe
# axle
```