NAME-ANUBHAV ANAND

ENROLLMENT NUMBER-2020CSB102

SUBJECT-ASSIGNMENT_1 OF ALGORITHM LAB

Q1.Construct large data sets taking random numbers from uniform distribution (UD)

Ans-Program-

```
In [16]:
import numpy as np
for i in range(6):
    x=np.random.uniform(low = 0, high = 5, size = pow(2,i))
    print(x)
```

```
In [2]: import math
import numpy as np
for i in range(6):
    x=np.random.normal(0,5, size = pow(2,i))
    print(x)
```

2-A: Implement Merge Sort (MS) and check for correctness.

```
In [3]: # Python program for implementation of MergeSort
def mergeSort(arr):
    if len(arr) > 1:

         # Finding the mid of the array
        mid = len(arr)//2

        # Dividing the array elements
        L = arr[:mid]

        # into 2 halves
        R = arr[mid:]

        # Sorting the first half
        mergeSort(L)

        # Sorting the second half
        mergeSort(R)

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
```

```
        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

# Code to print the list
```

-

```python
# Code to print the list


def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()


# Driver Code
if __name__ == '__main__':
    arr = [12, 11, 13, 5, 6, 7]
    print("Given array is", end="\n")
    printList(arr)
    mergeSort(arr)
    print("Sorted array is: ", end="\n")
    printList(arr)
```

RESULT:

```
Given array is
12 11 13 5 6 7
Sorted array is:
5 6 7 11 12 13
```

2.b Implement Quick Sort (QS) and check for correctness

```
In [5]: def partition(start, end, array):

            pivot_index = start
            pivot = array[pivot_index]

            while start < end:
                while start < len(array) and array[start] <= pivot:
                    start += 1

                while array[end] > pivot:
                    end -= 1

                if(start < end):
                    array[start], array[end] = array[end], array[start]

            array[end], array[pivot_index] = array[pivot_index], array[end]

            return end

        def quick_sort(start, end, array):

            if (start < end):

                p = partition(start, end, array)

                quick_sort(start, p - 1, array)
                quick_sort(p + 1, end, array)

        array = [ 10, 7, 8, 9, 1, 5 ]
        print(f'Given array: {array}')
        quick_sort(0, len(array) - 1, array)

        print(f'Sorted array: {array}')
```

RESULT:

```
Given array: [10, 7, 8, 9, 1, 5]
Sorted array: [1, 5, 7, 8, 9, 10]
```

Q3. Count the operations performed, like comparisons and swaps with problem size increasing in
powers of 2, for both MS and QS with both UD and ND as input data.
Ans-Program-For uniform distribution & Normal distribution.
Ans-Merge Sort-

Merge Sort

In [28]:

```
from time import time
import numpy as np
import math
def create(i):
    x=np.array(np.random.uniform(low = 1, high = 500, size = pow(2,i)))
    return x

operations=[]
# Python program for implementation of MergeSort
def mergeSort(arr):
    if len(arr) <= 1: return arr
     # Finding the mid of the array
    mid = len(arr)//2

    # Sorting the first half
    L = mergeSort(arr[:mid])

    # Sorting the second half
    R = mergeSort(arr[mid:])

    i = j = 0

    merged_arr = []
    # Copy data to temp arrays L[] and R[]
    c=0
```


```
    merged_arr = []
    # Copy data to temp arrays L[] and R[]
    c=0
```

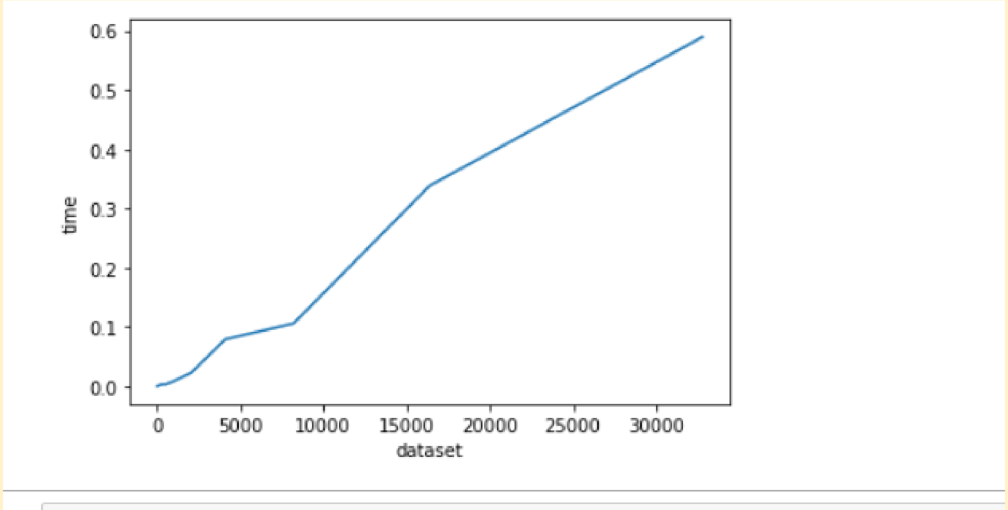

```
    while i < len(L) and j < len(R):
        if L[i] < R[j]:
            merged_arr.append(L[i])
            i += 1
            c+=1
        else:
            merged_arr.append(R[j])
            j += 1
            c+=1
    # Checking if any element was left
    while i < len(L):
        merged_arr.append(L[i])
        i += 1
        c+=1
    while j < len(R):
        merged_arr.append(R[j])
        j += 1
        c+=1
    operations.append(c)
    return merged_arr

# Code to print the list
```

```python
# Code to print the List


def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()


# Driver Code
final_list=[]
y=[]
for i in range(16):
    n=math.pow(2,i)
    arr = create(i)
    print("Given array is", end="\n")
    printList(arr)
    start=time()
    arr = mergeSort(arr)
    end=time()
    print("Sorted array is: ", end="\n")
    printList(arr)
    print(f"Execution time : {end - start} s")
    final_list.append(end - start)
    if n!=1:
        y.append(final_list[i]/(n*math.log(n,2)))
print(final_list)
```

Code to plot graph-

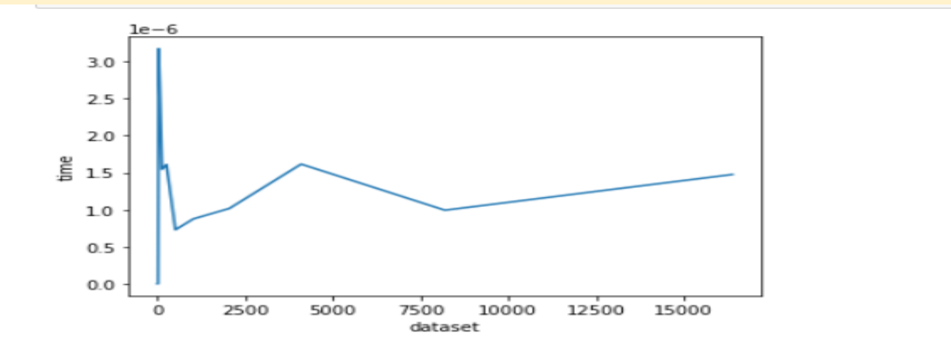
```python
In [29]: dataset=[]
         for i in range(16):
                 t=math.pow(2,i)
                 dataset.append(t)
         import matplotlib.pyplot as plt
         plt.xlabel("dataset")
         plt.ylabel("time")
         plt.plot(dataset,final_list)
         plt.show()
```

Graph(Without dividing actual time with estimated time, For Uniform distribution)-



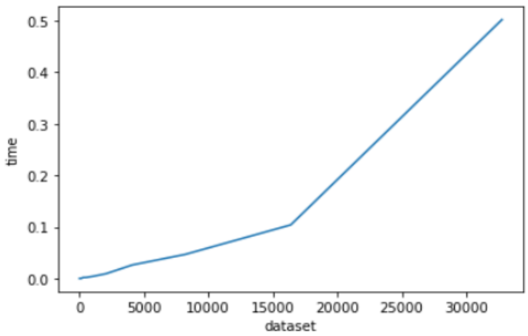
Code + graph(Dividing actual time with estimated time, Uniform distribution)-

```
In [30]: dataset=[]
         y=[]
         for i in range(15):
                 t=math.pow(2,i)
                 dataset.append(t)
                 if t!=1:
                     y.append(final_list[i]/(t*math.log(t,2)))


         import matplotlib.pyplot as plt
         dataset=dataset[1:]
         plt.xlabel("dataset")
         plt.ylabel("time")
         plt.plot(dataset,y)
         plt.show()
```



Graph(Without dividing actual time with estimated time, For Normal distribution)-

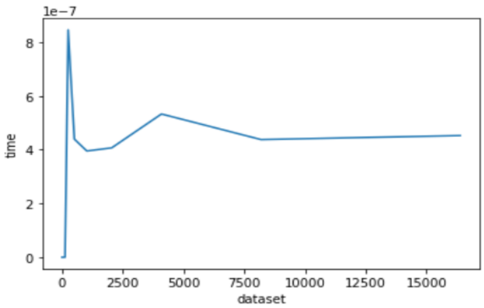
```
In [4]: dataset=[]
        for i in range(16):
                t=math.pow(2,i)
                dataset.append(t)
        import matplotlib.pyplot as plt
        plt.xlabel("dataset")
        plt.ylabel("time")
        plt.plot(dataset,final_list)
        plt.show()
```



Code + graph(Dividing actual time with estimated time, Normal distribution)-

In [5]:

```
dataset=[]
y=[]
for i in range(15):
        t=math.pow(2,i)
        dataset.append(t)
        if t!=1:
            y.append(final_list[i]/(t*math.log(t,2)))


import matplotlib.pyplot as plt
dataset=dataset[1:]
plt.xlabel("dataset")
plt.ylabel("time")
plt.plot(dataset,y)
plt.show()
```



Quick sort-

In [32]:

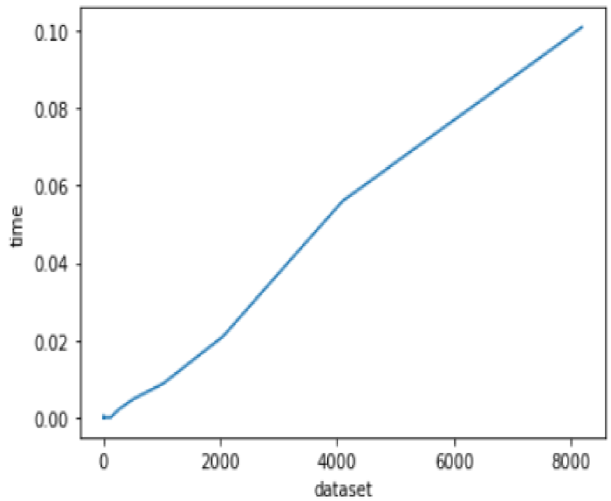
```
#  Python3 implementation of QuickSort
# This Function handles sorting part of quick sort
# start and end points to first and last element of
# an array respectively

def partition(start, end, array):

    # Initializing pivot's index to start
    pivot_index = start
    pivot = array[pivot_index]

    # This loop runs till start pointer crosses
    # end pointer, and when it does we swap the
    # pivot with element on end pointer
    while start < end:

        # Increment the start pointer till it finds an
        # element greater than  pivot
        while start < len(array) and array[start] <= pivot:
            start += 1

        # Decrement the end pointer till it finds an
        # element less than pivot
        while array[end] > pivot:
            end -= 1

        # If start and end have not crossed each other,
        # swap the numbers on start and end
        if(start < end):
            array[start], array[end] = array[end], array[start]

    # Swap pivot element with element on end pointer.
    # This puts pivot on its correct sorted place.
    array[end], array[pivot_index] = array[pivot_index], array[end]

    # Returning end pointer to divide the array into 2
    return end

# The main function that implements QuickSort
def quick_sort(start, end, array):

    if (start < end):

        # p is partitioning index, array[p]
```

```
def quick_sort(start, end, array):

    if (start < end):

        # p is partitioning index, array[p]
        # is at right place
        p = partition(start, end, array)

        # Sort elements before partition
        # and after partition
        quick_sort(start, p - 1, array)
        quick_sort(p + 1, end, array)

# Driver code
y=[]
time_list=[]
for i in range(14):
    n1=math.pow(2,i)
    array = create(i)
    print(array)
    start=time()
    quick_sort(0, len(array) - 1, array)
    end=time()
    print(f'Sorted array: {array}')
    print(f"Execution time : {end - start} s")
    time_list.append(end - start)
print(time_list)
```

Code + graph(Without Dividing actual time with estimated time, Uniform distribution)-

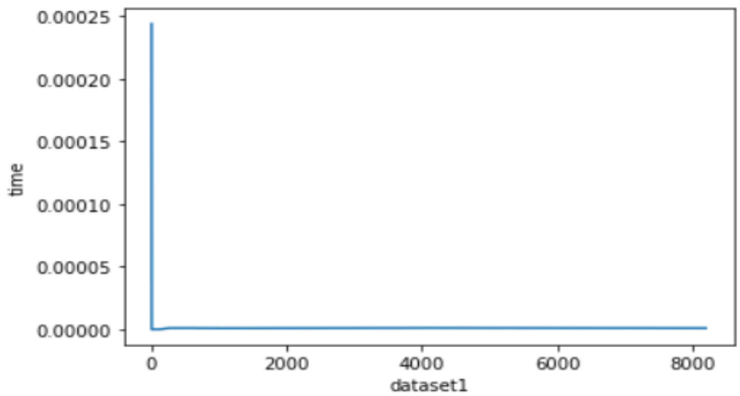
In [33]:

```
dataset=[]
for i in range(14):
        t=math.pow(2,i)
        dataset.append(t)
import matplotlib.pyplot as plt
plt.xlabel("dataset")
plt.ylabel("time")
plt.plot(dataset,time_list)
plt.show()
```



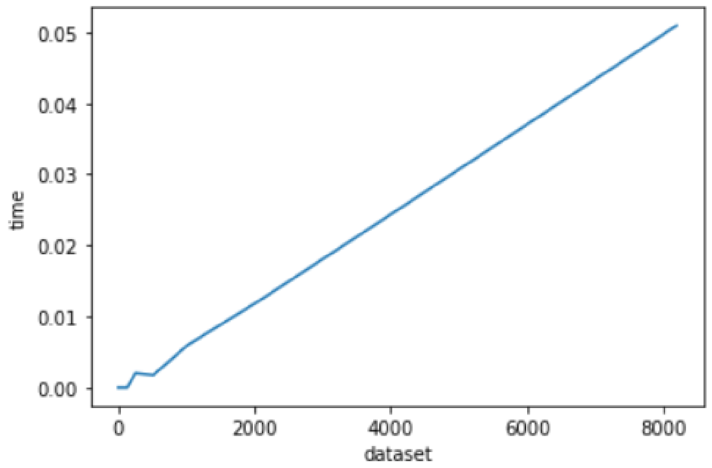
Code + graph(Dividing actual time with estimated time, Uniform distribution)-

In [34]:
```
dataset1=[]
y1=[]
for i in range(14):
        t=math.pow(2,i)
        dataset1.append(t)
        if t!=1:
            y1.append(time_list[i]/(t*math.log(t,2)))


import matplotlib.pyplot as plt
dataset1=dataset1[1:]
plt.xlabel("dataset1")
plt.ylabel("time")
plt.plot(dataset1,y1)
plt.show()
```



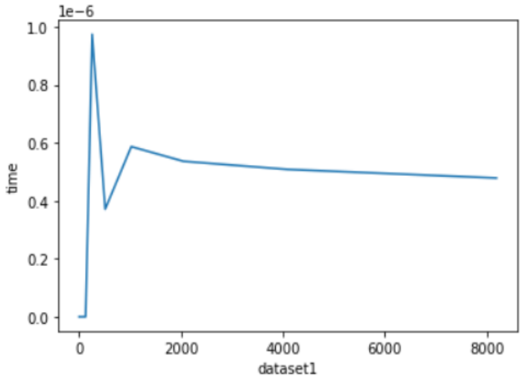
Code + graph(Without Dividing actual time with estimated time, Normal distribution)-

```
In [8]: dataset=[]
        for i in range(14):
                t=math.pow(2,i)
                dataset.append(t)
        import matplotlib.pyplot as plt
        plt.xlabel("dataset")
        plt.ylabel("time")
        plt.plot(dataset,time_list)
        plt.show()
```



Code + graph(With Dividing actual time with estimated time, Normal distribution)-

In [9]:
```
dataset1=[]
y1=[]
for i in range(14):
        t=math.pow(2,i)
        dataset1.append(t)
        if t!=1:
            y1.append(time_list[i]/(t*math.log(t,2)))


import matplotlib.pyplot as plt
dataset1=dataset1[1:]
plt.xlabel("dataset1")
plt.ylabel("time")
plt.plot(dataset1,y1)
plt.show()
```



4.Q Experiment with randomized QS (RQS) with both UD and ND as input data to arrive at the
average complexity (count of operations performed) with both input datasets.

Ans-Uniform distribution-

Randomised quick sort:-

35]:

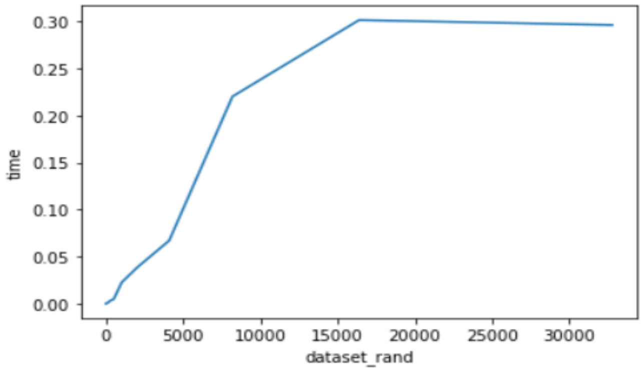
```
#  Python3 implementation of QuickSort
# This Function handles sorting part of quick sort
# start and end points to first and last element of
# an array respectively
import math
import numpy as np
def partition(start, end, array):

    # Initializing pivot's index to start
    pivot_index = np.random.randint(start,end)
    pivot = array[pivot_index]

    # This loop runs till start pointer crosses
    # end pointer, and when it does we swap the
    # pivot with element on end pointer
    while start < end:

        # Increment the start pointer till it finds an
        # element greater than  pivot
        while start < len(array) and array[start] <= pivot:
            start += 1

        # Decrement the end pointer till it finds an
        # element less than pivot
```

```
        # Decrement the end pointer till it finds an
        # element less than pivot
        while array[end] > pivot:
            end -= 1

        # If start and end have not crossed each other,
        # swap the numbers on start and end
        if(start < end):
            array[start], array[end] = array[end], array[start]

    # Swap pivot element with element on end pointer.
    # This puts pivot on its correct sorted place.
    array[end], array[pivot_index] = array[pivot_index], array[end]

    # Returning end pointer to divide the array into 2
    return end

# The main function that implements QuickSort
def quick_sort(start, end, array):

    if (start < end):

        # p is partitioning index, array[p]
        # is at right place
        p = partition(start, end, array)

        # Sort elements before partition
        # and after partition
        quick_sort(start, p - 1, array)
        quick_sort(p + 1, end, array)
```

```
time1_list=[]
average_list=[]
for i in range(16):
    n1=math.pow(2,i)
    array = create(i)
    print(array)
    k=12
    temp=0
    for i in range(k):
        start=time()
        quick_sort(0, len(array) - 1, array)
        end=time()
        temp+=(end-start)
    average_list.append(temp/k)
    time1_list.append(end - start)
    print(f'Sorted array: {array}')
    print(f"Execution time : {end - start} s")
print(time_list)
```

In [36]:

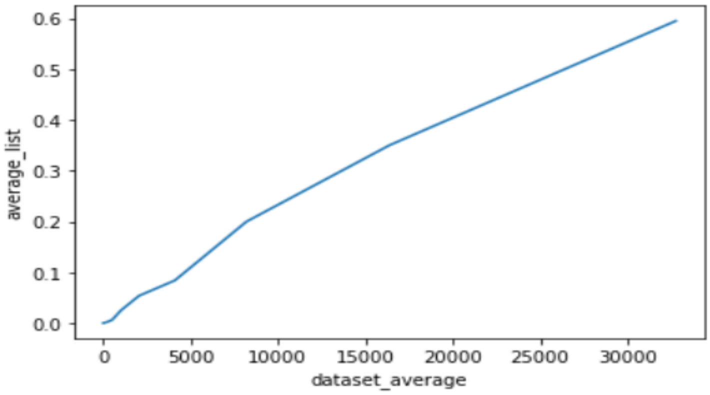
```
dataset_rand=[]
for i in range(16):
        t=math.pow(2,i)
        dataset_rand.append(t)
import matplotlib.pyplot as plt
plt.xlabel("dataset_rand")
plt.ylabel("time")
plt.plot(dataset_rand,time1_list)
plt.show()
```



average Complexity

In [48]:

```
dataset_average=[]
for i in range(16):
        t=math.pow(2,i)
        dataset_average.append(t)
import matplotlib.pyplot as plt
plt.xlabel("dataset_average")
plt.ylabel("average_list")
plt.plot(dataset_average,average_list)
plt.show()
```



Normal Distribution-

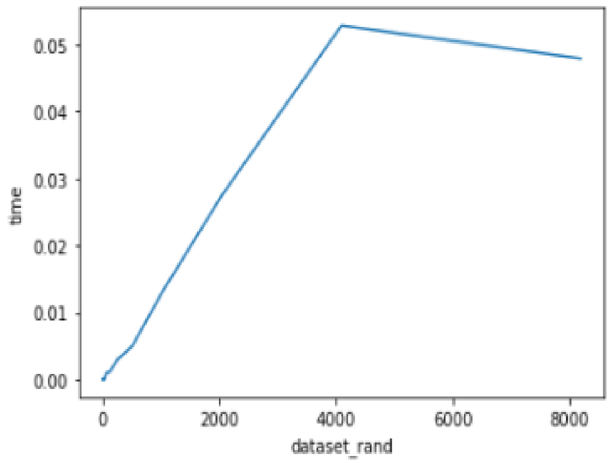
Randomised quick sort:-

In [10]:
```
#  Python3 implementation of QuickSort
# This Function handles sorting part of quick sort
# start and end points to first and last element of
# an array respectively
import numpy as np
def partition(start, end, array):

    # Initializing pivot's index to start
    pivot_index = np.random.randint(start,end)
    pivot = array[pivot_index]

    # This loop runs till start pointer crosses
    # end pointer, and when it does we swap the
    # pivot with element on end pointer
    while start < end:

        # Increment the start pointer till it finds an
        # element greater than  pivot
        while start < len(array) and array[start] <= pivot:
            start += 1

        # Decrement the end pointer till it finds an
        # element less than pivot
        while array[end] > pivot:
            end -= 1

        # If start and end have not crossed each other,
        # swap the numbers on start and end
        if(start < end):
            array[start], array[end] = array[end], array[start]

    # Swap pivot element with element on end pointer.
    # This puts pivot on its correct sorted place.
    array[end], array[pivot_index] = array[pivot_index], array[end]
```

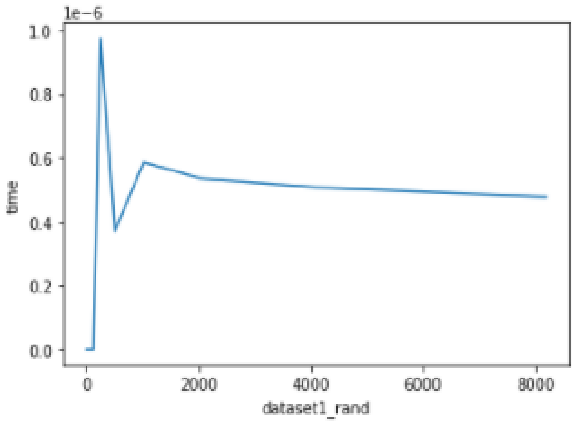
```python
    # Returning end pointer to divide the array into 2
    return end

# The main function that implements QuickSort
def quick_sort(start, end, array):

    if (start < end):

        # p is partitioning index, array[p]
        # is at right place
        p = partition(start, end, array)

        # Sort elements before partition
        # and after partition
        quick_sort(start, p - 1, array)
        quick_sort(p + 1, end, array)

# Driver code
time1_list=[]
average_list=[]
for i in range(14):
    n1=math.pow(2,i)
    array = create(i)
    print(array)
    k=12
    temp=0
    for i in range(k):
        start=time()
        quick_sort(0, len(array) - 1, array)
        end=time()
        temp+=(end-start)
    average_list.append(temp/k)
    time1_list.append(end - start)
    print(f'Sorted array: {array}')
    print(f"Execution time : {end - start} s")
print(time_list)
```

In [11]:

```python
dataset_rand=[]
for i in range(14):
        t=math.pow(2,i)
        dataset_rand.append(t)
import matplotlib.pyplot as plt
plt.xlabel("dataset_rand")
plt.ylabel("time")
plt.plot(dataset_rand,time1_list)
plt.show()
```

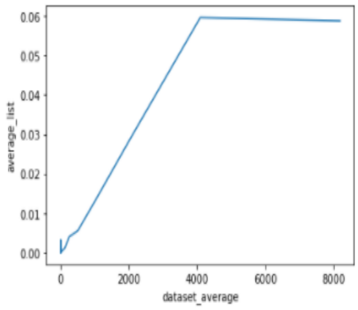


```
[12]: dataset1_rand=[]
y2=[]
for i in range(14):
        t=math.pow(2,i)
        dataset1_rand.append(t)
        if t!=1:
            y2.append(time_list[i]/(t*math.log(t,2)))


import matplotlib.pyplot as plt
dataset1_rand=dataset1_rand[1:]
plt.xlabel("dataset1_rand")
plt.ylabel("time")
plt.plot(dataset1_rand,y2)
plt.show()
```



average Complexity

```
In [13]: dataset_average=[]
for i in range(14):
        t=math.pow(2,i)
        dataset_average.append(t)
import matplotlib.pyplot as plt
plt.xlabel("dataset_average")
plt.ylabel("average_list")
plt.plot(dataset_average,average_list)
plt.show()
```



5. Now normalize both the datasets in the range from 0 to 1 and implement bucket sort (BS)
algorithm and check for correctness.

Ans-

✓ UD

```
[2]: from time import time
     import numpy as np
     import math
     def create3(i):
         x=np.array(np.random.uniform(low=0,high=1,size=pow(2,i)))
         return x
     def insertionSort(b):
         for i in range(1, len(b)):
             up = b[i]
             j = i - 1
             while j >= 0 and b[j] > up:
                 b[j + 1] = b[j]
                 j -= 1
             b[j + 1] = up
         return b
```

```
def bucketSort(x):
    arr = []
    slot_num = 10
    for i in range(slot_num):
        arr.append([])

    for j in x:
        index_b = int(slot_num * j)
        arr[index_b].append(j)

    for i in range(slot_num):
        arr[i] = insertionSort(arr[i])

    k = 0
```

```
    k = 0
    for i in range(slot_num):
        for j in range(len(arr[i])):
            x[k] = arr[i][j]
            k += 1
    return x
x =create3(3)
print(x)
print("Sorted Array is")
print(bucketSort(x))
```

```
[0.56449324 0.834125   0.65253786 0.98143797 0.68898788 0.36968947
 0.50722634 0.67353586]
Sorted Array is
[0.36968947 0.50722634 0.56449324 0.65253786 0.67353586 0.68898788
 0.834125   0.98143797]
```

✓ ND

```
In [11]: from time import time
         import numpy as np
         import math
         def create1(i):
             x=np.array(np.random.normal(0.5,0.1, size = pow(2,i)))
             return x
         def insertionSort(b):
             for i in range(1, len(b)):
                 up = b[i]
                 j = i - 1
                 while j >= 0 and b[j] > up:
                     b[j + 1] = b[j]
                     j -= 1
                 b[j + 1] = up
             return b
```

6.Experiment with BS to arrive at its average complexity for both UD and ND data sets and infer.

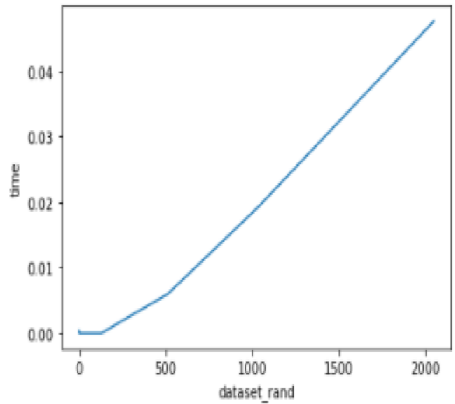
For Unifor distribution-

Bucket Sort

```
In [39]: # Python3 program to sort an array
         # using bucket sort
         from time import time
         import numpy as np
         import math
         def create1(i):
             x=np.array(np.random.uniform(low = 0, high = 1, size = pow(2,i)))
             return x
         def insertionSort(b):
             for i in range(1, len(b)):
                 up = b[i]
                 j = i - 1
                 while j >= 0 and b[j] > up:
                     b[j + 1] = b[j]
                     j -= 1
                 b[j + 1] = up
             return b

         def bucketSort(x):
             arr = []
             slot_num = 10 # 10 means 10 slots, each
                           # slot's size is 0.1
             for i in range(slot_num):
                 arr.append([])

             # Put array elements in different buckets
             for j in x:
                 index_b = int(slot_num * j)
                 arr[index_b].append(j)

             # Sort individual buckets
             for i in range(slot_num):
                 arr[i] = insertionSort(arr[i])

             # concatenate the result
             k = 0
             for i in range(slot_num):
                 for j in range(len(arr[i])):
                     x[k] = arr[i][j]
                     k += 1
             return x

         # Driver Code
         # x = [0.897, 0.565, 0.656,
```

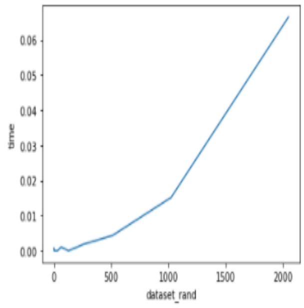
```python
# Driver Code
# x = [0.897, 0.565, 0.656,
#      0.1234, 0.665, 0.3434]
# print("Sorted Array is")
# print(bucketSort(x))
time2_list=[]
for i in range(12):
    array=create1(i)
    start=time()
    bucketSort(array)
    end=time()
    time2_list.append(end-start)
```

In [40]:
```python
dataset_rand=[]
for i in range(12):
        t=math.pow(2,i)
        dataset_rand.append(t)
import matplotlib.pyplot as plt
plt.xlabel("dataset_rand")
plt.ylabel("time")
plt.plot(dataset_rand,time2_list)
plt.show()
```



For Normal Distribution-

```
In [15]: dataset_rand=[]
         for i in range(12):
                 t=math.pow(2,i)
                 dataset_rand.append(t)
         import matplotlib.pyplot as plt
         plt.xlabel("dataset_rand")
         plt.ylabel("time")
         plt.plot(dataset_rand,time2_list)
         plt.show()
```



7. Implement the worst case linear median selection algorithm by taking the median of medians
(MoM) as the pivotal element and check for correctness.

Ans-

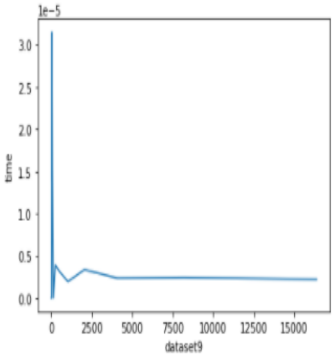
Medians of medians

```
In [44]: import numpy as np
         from time import time
         import math

         def create(i):
             x=np.array(np.random.uniform(low=1,high=1000,size=pow(2,i)))
             return x


         def insertionsort(arr,initial,final):
             for i in range(initial,final+1):
                 value=arr[i]
                 pos=i-1
                 while pos>=initial and arr[pos]>value:
                     arr[pos+1]=arr[pos]
                     pos-=1
```

```
        pos=i-1
        while pos>=initial and arr[pos]>value:
            arr[pos+1]=arr[pos]
            pos-=1



def getmedian(arr,initial,final):
    insertionsort(arr,initial,final)
    return arr[int((initial+final)/2)];



def median_of_median(arr,arrSize,divideSize):
    if arrSize < divideSize:
        return getmedian(arr,0,arrSize-1)


    fullgroup=int(arrSize/divideSize)
    elements_in_last=arrSize%divideSize

    if(elements_in_last==0):
        newarrSize=fullgroup
    else:
        newarrSize=fullgroup+1

    newarr=[]

    for i in range(newarrSize):
        if i==newarrSize-1:
            newarr.append(getmedian(arr,(divideSize*i),arrSize-1))
        else:
            newarr.append(getmedian(arr,(divideSize*i),(divideSize*(i+1)-1)))
    return median_of_median(newarr,newarrSize,divideSize)
```

```
array=[]
time0=[]
dataset=[]

for i in range(15):
    n1=math.pow(2,i)
    dataset.append(n1)
    array=create(i)
    print(array)
    start=time()
    median=median_of_median(array,len(array),5)
    end=time()
    print(median)
    time0.append((end-start)/n1)

print(time0)

import matplotlib.pyplot as plt
plt.xlabel("dataset")
plt.ylabel("time")
plt.plot(dataset,time0)
plt.show()
```

```
In [45]: dataset9=[]
         for i in range(15):
             n1=math.pow(2,i)
             dataset9.append(n1)

         import matplotlib.pyplot as plt
         plt.xlabel("dataset9")
         plt.ylabel("time")
         plt.plot(dataset9,time0)
         plt.show()
```

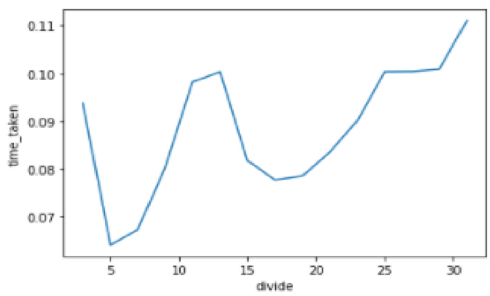


8. Take different sizes for each trivial partition (3/5/7) and see how the time taken is changing.

Ans-

Medians of medians with size 3,5,7

```
In [46]: array1=create(15)
         divide=[]
         time_taken=[]
         for i in range(15):
             divide.append(2*i+3)
             start=time()
             median=median_of_median(array1,len(array1),2*i+3)
             end=time()
             time_taken.append((end-start))
         import matplotlib.pyplot as plt
         plt.xlabel("divide")
         plt.ylabel("time_taken")
         plt.plot(divide,time_taken)
         plt.show()
```



9. Perform experiments by rearranging the elements of the datasets (both UD and ND) and
comment on the partition or split obtained using the pivotal element chosen as MoM.

Ans-

In [47]:
```
def partition(arr,low,high):
    pivot=arr[high]
    it=low-1
    for j in range(low,high+1):
        if arr[j]<pivot:
            it+=1
            arr[it],arr[j]=arr[j],arr[it]

    arr[it+1],arr[high]=arr[high],arr[it+1]
    return it+1


def findpartition(arr,arrSize,divideSize):
    val=median_of_median(arr,arrSize,divideSize)
    for i in range(arrSize):
        if arr[i]==val:
            arr[arrSize-1],arr[i]=arr[i],arr[arrSize-1]
    return partition(arr,0,arrSize-1)
array3=[]
dataset3=[]
partition3=[]
for i in range(15):
    n3=math.pow(2,i)
```

```
        arr[arrSize-1],arr[i]=arr[i],arr[arrSize-1]
    return partition(arr,0,arrSize-1)
array3=[]
dataset3=[]
partition3=[]
for i in range(15):
    n3=math.pow(2,i)
    array3=create(i)
    dataset3.append(math.pow(2,i))
    partition3.append((findpartition(array3,len(array3),5))/n3)



import matplotlib.pyplot as plt
plt.xlabel("dataset")
plt.ylabel("partition/n")
plt.plot(dataset3,partition3)
plt.show()
```