Randomised Quicksort Algorithm

Submitted by Daiyan Khan (19141024)

What is quicksort?

We know that a quicksort algorithm is an efficient sorting algorithm that partitions an array into smaller arrays after a pivot is selected (usually the first, last, or median element of the array). It calls itself recursively to sort the subsequent arrays that are formed in relation to the chosen pivot value. The values less than the pivot are set to the left of the pivot, and the values larger than the pivot are set to its right. This continues until the whole array is sorted.

What is Randomised Quicksort?

Randomised quicksort is when a random value from the array is chosen as a pivot to sort the array using the quicksort algorithm.

Code Summary

1. The code will first take a user input to define the range 'r' and length 's' that will be used to create a randomised array of length s that consists of unique integers between the range 0 to r (no repetition of integer values).

2. It will then pick a random value from the newly created unsorted array to be used as a pivot for quicksorting. The initial array is split into two seperate arrays that will separately hold values that are less than and greater than the chosen pivot value, respectively.

3. Quicksort will be applied on the splitted arrays and it will continue recursively calling itself till the entire original array is sorted. Meanwhile printing how many comparisons were required to sort each splitted array using the chosen pivot.

3. Finally, will print the sorted array and the total number of comparisons required to sort the array entirely.

Random Number Generator

A random number generator 'rng' is set up in the driver code to create an array that consists of a user defined length, with a range of integer values starting 0 to n.

Code Description

The code below consists of three (3) main methods:

- quicksort( )
- rand( )
- partition( )

quicksort( )

- This method is created to implement the quicksort algorithm. It takes 3 inputs: an array 'a', a variable to store the starting index value of the array 'i', and a variable to store the ending index value of the array 'j'.
- A pivot index value 'p_ind' is set to find the index value of the pivot in the array

- The original array is split into two arrays.

- The array is first partially sorted around the chosen pivot, with values < pivot and values > pivot being separated and stored in the two split arrays.

rand( )

- This method is used to choose a random number from the array to use as a pivot.

- This method then swaps the chosen pivot value with the first index value of the array to begin the quicksort algorithm.

partition( )

- This method carries out the sorting. As the rand( ) method has already set the first value in the array to be the pivot value, the partition( ) method carries out a quicksort algorithm as usual.

- A variable 'x' is set up to record where the partition begins in the array. A for loop runs from the beginning till the end of the array. If the current value in the loop is less than or equal to the pivot value, it is shifted to the left of the pivot.

- Else (if the value is greater than pivot) it is left as is, which will always be on the right of the pivot as the pivot has been set as the first value of the array, and the values shifted to the left of the array are only those that are smaller than or equal to the pivot.

- For every pivot chosen, the number of comparisons made between values in the array after using the chosen pivot will be printed.

--- CODE IN NEXT PAGE ---

CODE

```
# Import neccessary libraries and setup global variables

import time
import random
import numpy as np
from numpy.random import default_rng

'''
A global variable 'totalCom' is set up to keep
track of total number of comparisons made to
sort the array using quicksort.
'''
totalCom = 0
```

```
'''
Quicksort Algorithm:
Method is defined to implement quicksort algorithm
where it takes 3 inputs = array (a), starting index
of array (i), ending index of array (j)
'''

def quicksort(a, i, j):
    if i < j:

      # index position of the pivot relative to the array
        p_ind = rand(a, i, j)

      # Array is split into two halves, and theLeft and
      # right halves of the array are sorted seperately,
      # giving us a partially sorted array
        quicksort(a, i, p_ind - 1)
        quicksort(a, p_ind + 1, j)
```

```
# Random Pivot Generator:
# Method to pick a random pivot value, and set as
# first element of array to sort
def rand(a, i, j):

  # Pick a random number between array range i to j
    pivot = random.randrange(i, j)

  # swap pivot value with current first index value
  # and call partition method
    (a[i], a[pivot]) = (a[pivot], a[i])
    return partition(a, i, j)
```

```
'''
Partition:
Method to partition arrays. Values smaller than chosen
pivot is stored in the array on the left of the pivot,
and values larger than pivot are stored on its right.
Total number of comparisons is incremented for every
comparison made between array values.
'''

def partition(a, i, j):
    global totalCom
    pivot = i
    print('Pivot = ', a[pivot]) # print chosen pivot value
    count = 0
    x = i + 1 # array start position

    # if current index value is less than or equal
    # to pivot value, then shift to left of partition
    for j in range(i + 1, j + 1):
        if a[j] <= a[pivot]:
            (a[x], a[j]) = (a[j], a[x])
            x = x + 1
            count += 1
```

```
    # print the number of comparisons
    # every time a new pivot is picked
    print ('No. of comparisons:', count)
    totalCom += count
    (a[pivot], a[x - 1]) = (a[x - 1], a[pivot])
    pivot = x - 1
    return (pivot)
```

######### DRIVER CODE #########

```
'''
Random Number Array generator:
Generate an array of unique random numbers
between range 0 to n-1, array length = size,
set replace = False to get unique values
'''

print('# Random Array Generator # \n')
rng = default_rng()
n = 1000
s = 1001
array = rng.choice(n+1, size=s, replace=False)
print('Unsorted array:\n',array)
```

```
# Begin quicksort algorithm
quicksort(array, 0, len(array) - 1)
print('\n')
print('Sorted array: ',array)
print('Total no. of comparisons: ',totalCom)
```

```
'''
 Here a loop is implemented to run the sorting algorithm
 100 times on the unsorted array we created above. The
 number of comparisons made every time the algorithm is
 run is recorded. Then we calculate the average number
 of comparisons made in the 100 times the code was run.
'''

print('\n **** Start Quicksort ****')
time.sleep(1)

nr = 100 # number of time quicksort is run
sumCom = 0 # sum of total no. of comparisons

for i in range(0,nr):
  quicksort(array, 0, len(array) - 1)
  sumCom += totalCom # increment sum after every loop
  print('Total no. of comparisons: ',totalCom)
  print('---- Quicksort End ----')
  totalCom = 0 # reset comparison counter
```

```
# print sorted array and average num. of comparisons
print('Sorted array: ',array)
avg = sumCom / nr # calculate average number of comparisons for n number
of runs
print('Avg. num. of comparisons per loop = ',avg)
```

P.T.O

CALCULATING EXPECTED VALUE OF COMPARISONS FOR RANDOMISED QUICKSORT

Assume that for an unsorted array of 'n' numbers, A[n], the ith smallest value in the array is x_i, where no 2 values are equal.

→ This implies that sorted array A,

$$A[n] \text{ is } \{x_1, x_2, x_3 \dots x_n\}$$

→ Let $Z_{ij} = \{x_i \dots x_j\}$ denote all elements between and including x_i and x_j.

→ $X_{ij} = I\{x_i \text{ compared to } x_j\}$, where $X_{ij} = 1$ if compared or 0 if no comparison

∴ X_{ij} can be used to indicate the random variable for two elements from the unsorted array being compared when the quicksort is run.

∴ we can assume that the total number of comparisons made every time quicksort is run is denoted by -

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

Here we assume no two elements are compared twice.

→ We can calculate the expected number of comparisons to be -

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

where $E[X_{ij}] = P(x_i \text{ compared to } x_j)$

→ The probability of comparison can be calculated by,

$P(x_i \text{ or } x_j \text{ is the first pivot}) = P(x_i \text{ is the first pivot}) + P(x_j \text{ is the first pivot})$

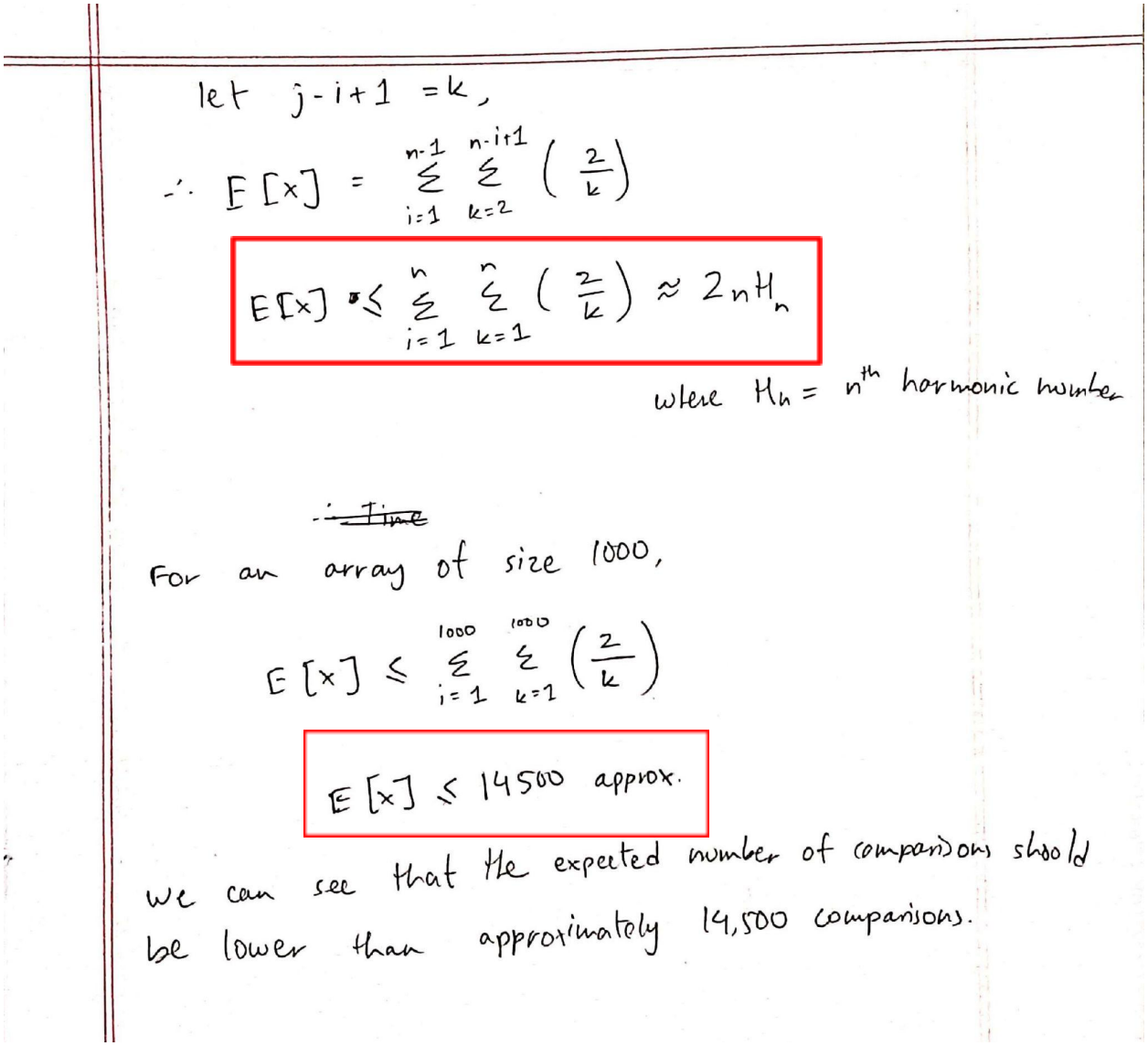
since there are a total of $j-i+1$ numbers,

$$P(x_i \text{ or } x_j) = \frac{1}{j-i+1} + \frac{1}{j-i+1}$$

$$= \frac{2}{j-i+1}$$

$$\therefore E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(\frac{2}{j-i+1} \right)$$

PTO



From the above calculations we can see that E[X] (average number of comparisons made) of our code must be lower than the calculated E[X] of 14,500 to be a feasible code.

```
1 # print sorted array and average num. of comparisons
2
3 print('Sorted array: ',array)
4 avg = sumCom / nr # calculate average number of comparisons
5 print('Avg. num. of comparisons per loop = ',avg)
```

```
Sorted array:  [   0    1    2 ...  998  999 1000]
Avg. num. of comparisons per loop =  5069.75
```

We can see that the code has achieved an expected value of approximately 5070 comparisons after being run 100 times, which is well below our expected E[X] of a randomised quicksort.