LTCE

Experiment No. 3

Quick Sort

Aim - Implementation of quick sort.

Theory -

Algo -

1. Make the right-most index value pivot.
2. Partition the array using pilot value.
3. Quicksort left partition recursively.
4. Quicksoot right partition recursively.

Example -

Consider the array : 50, 23, 9, 18, 61, 32

1. Make an element as pilot
   - In our case low is 0 & high is 5
   - Values at low & high are 50 & 32 & value of pilot is 32
2. Partiation the array on the basis of pilot.
   - In the partiation function, we start from the 1st element & compare it with pivot
   - Since 50 is greater than 32, we don't make any change & move on to the next element 23
   - Compare again with pivot. The array becomes 23, 50, 9, 18, 61, 32
   - We move onto the next element 9 which is less than 32, the array becomes 23, 9, 18, 50, 61, 32
   - Lastly, we swap our pivot with 50 so that it comes to the correct position.
3. The main array after the 1st step becomes 23, 9, 18, 32, 61, 50
4. Now list is divided into 2 parts.

a) Sublist before pivot element

LTCE

b) Sublist after pivot element.

5. Repeat the steps for left & right recurrsively. The final array thus becomes 9, 18, 23, 32, 50, 61

Time complexity -

Best	Average	Wrost
$\Omega(n\log(n))$	$\Theta(n\log(n))$	$O(n\log(n))$

Conclusion -

Hence, we have successfully implemented algo & analysed the complexity.

14/15

Python program to parform quicksort

```
def partition (array,low ,high):
   pivot=array[high]

   i=low-1
   for j in range(low ,high):
      if array[j]<= pivot:
         i=i+1
         array[i],array[j]= array[j],array[i]
   array[i+1],array[high]= array[high],array[i+1]
   return i+1

def quickSort(array,low,high):
   if low <high:
      pi=partition(array,low,high)
      quickSort(array,low,pi-1)
      quickSort(array,pi+1,high)

n=int(input("Enter number of elements in list"))
arr=[]
for i in range(n):
   b=int(input())
   arr.append(b)

print("Given array is ",arr)
quickSort(arr,0,n-1)
print("Sorted array is ",arr)
```

```
/bin/python3 "/home/computer/Downloads/SIMPLE_INVENTORY_SYSTEM_IN_PYTHON_WITH_SOURCE_CODE/Simple_Inventory PYTHON/Simple Inventory System/QuickSort.py"
● (base) computer@computer-ThinkCentre:~/Downloads/SIMPLE_INVENTORY_SYSTEM_IN_PYTHON_WITH_SOURCE_CODE/Simple_Inventory PYTHON/Simple Inventory System$ /bin/python3 "/home/com
puter/Downloads/SIMPLE_INVENTORY_SYSTEM_IN_PYTHON_WITH_SOURCE_CODE/Simple_Inventory PYTHON/Simple Inventory System/QuickSort.py"
Enter number of elements in list5
23
45
67
34
78
Given array is  [23, 45, 67, 34, 78]
Sorted array is  [23, 34, 45, 67, 78]
○ (base) computer@computer-ThinkCentre:~/Downloads/SIMPLE_INVENTORY_SYSTEM_IN_PYTHON_WITH_SOURCE_CODE/Simple_Inventory PYTHON/Simple Inventory System$ ▌
```

Experiment No. 4
Fractional Knapsack Problem

Aim - To implement fractional knapsack problem.

Theory -

Algo -

1. Node root represents the initial state of the knapsack, where you have not selected any package.
   - Total value = 0. The upper bound of the root node = M*M -aximum unit cost.
2. Node root will have child nodes corresponding to the ability to select the package with the largest unit-cost. For each node, you re-cal-culate the parameters.
3. In child nodes, you will prioritize branching for the node having the larger upper-bond. The children of this node corresponds to the ability of selecting the next package having large unit cost.
4. Repeat step 3 with the note - for nodes with upper bou is lower or equals to the temporary maximum cost of an option found you do not need to branch for that node.
5. If all nodes are branched or cut off, the most expensiv option is the one to look for.

~~Alg~~ Example -

- The parameters of the problem are $n = 3$; $M = 11$.
- The packages: { i = i; w[i] = 5; r[i] = 10; { i = 2; w[i] ; v[i] = 16 }; { i = 3; w[i] = 10; v[i] = 28 } -> Light weight but the values is also very light.
- The algo will select (package 1, package 2) with a tot value of 26, while the optimal soln of the probler

is (package 3) with a total value of 28.

Conclusion -

Hence, we have successfully implemented algo.

14/15

Activities Visual Studio Code ▾ Tue 1:48 PM ●

● anuj.py - VidZY-master - Visual Studio Code

File Edit Selection View Go Run Terminal Help

alogarithm.py | job.py | Untitled-1 | anuj.py ● | quick_sort.py

home > computer > Desktop > anuj.py > ...

```python
def knapSack(W, wt, val, n):

    if n == 0 or W == 0 :
        return 0

    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1)

    else:
        return max(val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
            knapSack(W, wt, val, n-1))

val = [50,100,150,200]
wt = [8,16,32,40]
W = 64
n = len(val)
print (knapSack(W, wt, val, n))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
/usr/bin/python3.9 /home/computer/Desktop/anuj.py
(base) computer@computer:~/Downloads/VidZY-master$ /usr/bin/python3.9 /home/computer/Desktop/anuj.py
350
(base) computer@computer:~/Downloads/VidZY-master$
```

⊗ 0 ⚠ 0 Ln 14, Col 18 Spaces: 4 UTF-8 LF {} Python 3.9.12 64-bit