Design And Analysis Of Algorithms-Assignment 8
Group-2

ADITYA RAJ(IIT2017005)
MAYANK MRINAL(IIT2017006)
MOHIT PAHUJA(IIT2017007)
ARASHPREET SINGH (IIT2017008)

***Abstract*— This document is complete design and analysis of algorithm ,optimal fractional knapsack filling .We have used greedy approach for the above problem.**

I. INTRODUCTION

For this problem ,we have implemented greedy algorithm through C++. Filling of knapsack is done based on the ratio of profit to weight of the elements

This report further contains::
II. Algorithm Design.
III.Algorithm Analysis
IV.Experimental Study.
V.Conclusions.
VI.References.

II. ALGORITHM DESIGN

We take input of the profits and weights of all the elements available to fill.Let the number of elements available be n. We have created a structure which contains index i and ratio r of profit by weight of each element. We have created an array of the structure for all the elements named P. Now we have used heap sort this array. Now we have initialized two variables for current weight of knapsack to 0 and an index which will iterate to the elements of the structure array to n. Now in a while loop we have iterated to all the element of the array P.If the weight of the element at index specified by the array P is less than the remaining weight of knapsack, we add this element to the knapsack completely else we add a fraction of that element to the knapsack .

Algorithm 1

define n←10000
struct R()
int i
float r
main()
input←n,w

for(int i=0;i¡n;i++) cin¿¿wt[i]¿¿val[i]; ppw[i].i=i; ppw[i].r=((float)val[i])/((float)wt[i]); **while** $n \geq 0$ **do**
  if *step!=rev* **then**
    sum<sum+step
  else
  end
  sum←sum- step
  v.push_back(sum)
  step++
end
heapsort(ppw)
float curr_wt←0.0
float ans←0
int idx←n-1
while $curr_wt \leq widx \geq 0$ **do**
  if $wt[ppw[idx].i] \leq w - curr_wt$ **then**
    $ans \leftarrow ans + val[ppw[idx].i]$
    $curr\_wt \leftarrow curr\_wt + wt[ppw[idx].i]$
  else
    $ans \leftarrow ans + (ppw[idx].r) * (w - curr\_wt)$
    $curr\_wt \leftarrow w$
  end
  idx–
end
print←ans

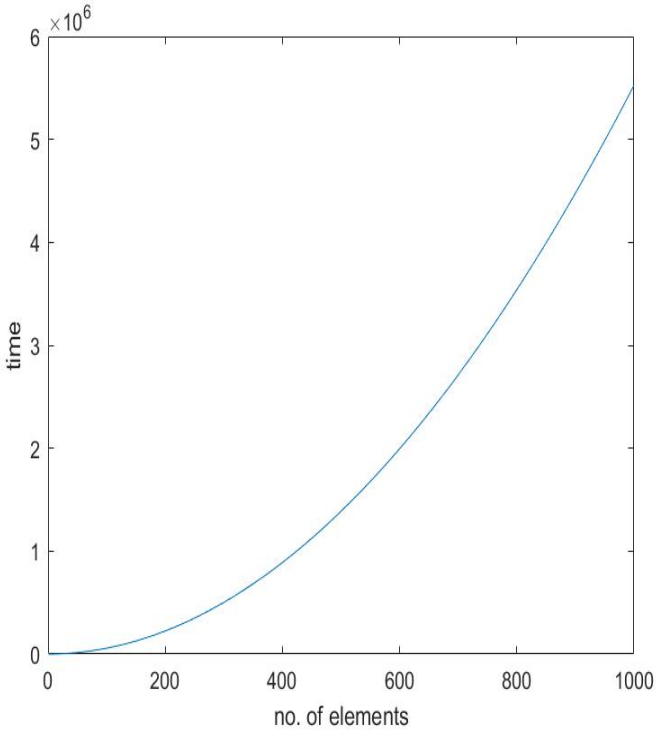
III. ALGORITHM ANALYSIS

A. Time Complexity

- *The time complexity of our algorithm mainly depends on time taken to sort array of structures plus the time taken to traverse that array.*
- *Heapsort is used for sorting the array which takes O(nlog(n)) in worst case.Therefore the worst case time complexity is O(nlog(n)) +O(n) which is the time taken to traverse that array ,which is equal to O(nlog(n)).*
- *The best case time complexity is encountered when the array which store the ratios is already sorted . In this case heapsort takes O(n).Therefore the best case complexity is O(n).*

B. Space Complexity

- *There are two arrays of size n which stores the weight and values of the available elements for filling. Also we have created an array of structures of size n which has two elements in it.Therefore the total space complexity is O(4n) i.e. O(n)*



The graph depicts that the time calculated i.e.O(nlog(n)).

IV. EXPERIMENTAL STUDY

The graph depicts that the time calculated i.e.O(nlog(n)).

The integrated Development environment of C++ is used for processing the algorithm and graphical analysis is done using MATLAB plot function.

V. CONCLUSIONS

In this document we conclude that our algorithm has worst case time complexity is O(nlog(n)) +O(n) and the best case complexity is O(n).

VI. REFERENCES

Previous assignments.
THOMAS H. CORMEN.