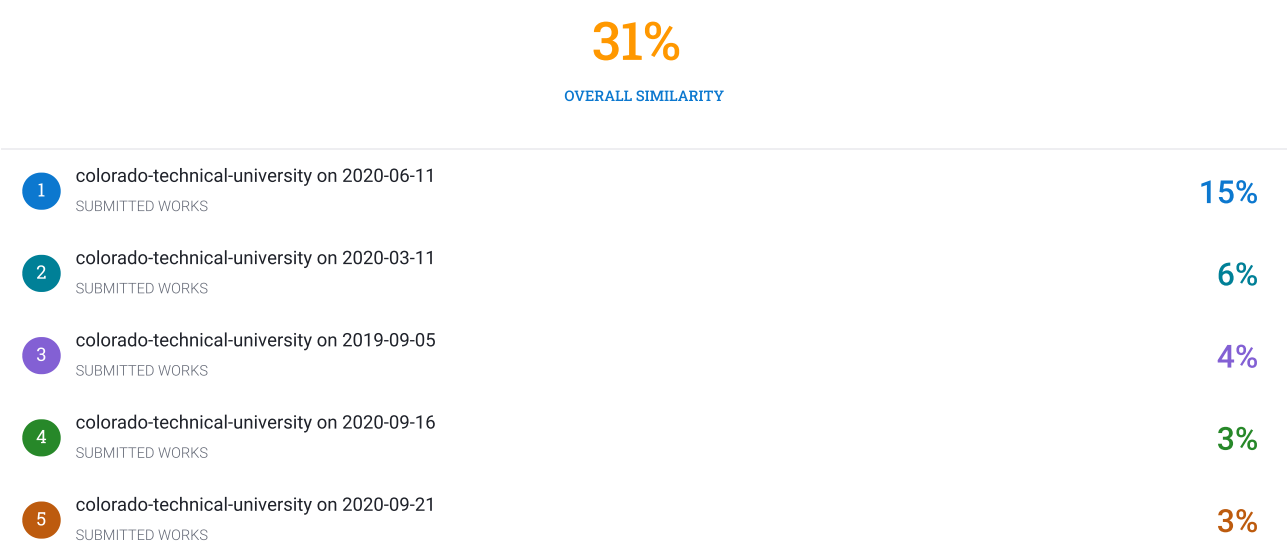


# ValarianCouch\_CS627\_IP3.docx

## Sources Overview



### Excluded search repositories:

- None

### Excluded from Similarity Report:

- Bibliography
- Quotes
- Small Matches (less than 8 words).

### Excluded sources:

- None

**Part 1:**

Considering the following as the set of given inputs:

$n$  being the files number

$s[n]$  is the array which has the independent file size.

$m$  is <sup>4</sup> disks

$t[m]$  is the storage on each disk

The following <sup>5</sup> is the algorithm/pseudocode used for the transfer of  $n$  files to the  $m$  disks while storing the results on storage on  $map[i]$  which is the disk index.

Function GreedyAlgoFit (<sup>1</sup>  $n, m, s[], t[]$ )

1. Declare map array of size  $n$ .
2. Declare file array  $file\_array[]$  of size  $n$  //Stores indices of original files
3. Create disk array  $disk\_array[]$  of size  $m$  //stores indeces of original disks
4. <sup>3</sup> Sort the files in descending order
5. Sort the disks in descending order
6. <sup>2</sup> For  $k=0$  to  $n$  and  $l=0$  to  $m$ , execute:
  - I. If  $t[k] \geq s[l]$
  - II.  $Map[file\_array[k]] = disk\_array[l]$
  - III.  $t[l] = t[l] - s[k]$
  - IV. <sup>1</sup> Increment the variables
7. Return map.

**Part 2:**

My solution provided above is implemented based on greedy algorithm. It therefore can't guarantee an optimal solution for both cases.

Estimating the running time:

- 1 and 2 steps takes  $O(n+m)$  which is  $O(n \log n)$  same as merge sort in merging the arrays
- 3 to 4 takes  $O(m+n)$  which is same as  $O(n)$  time
- The other steps takes a constant time.
- <sup>1</sup>  $T(n) = O(n \log n) + O(m \log m) + O(m+n)$ .

**Part 3:**

The algorithm will result to  $O(m*n)$  running time estimation in terms of big Oh. This is because brute force compares <sup>1</sup> the size of every file in every space of all the disks.