# Analysis for Greedy Approach

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Items** | **Capacity** | **Sequential** | **2 Threads** | **4 Threads** | **8 Threads** | **16 Threads** |
| **100** | **100** | **0** | **0** | **0** | **0** | **0** |
| **100** | **10000** | **15479** | **0** | **0** | **0** | **0** |
| **100** | **1000000** | **0** | **0** | **0** | **0** | **0** |
| **10000** | **100** | **533835** | **146** | **124** | **123** | **132** |
| **1000000** | **100** | **81907478** | **13271** | **14906** | **14491** | **17071** |

1. The only scope for parallelization in greedy approach for knapsack is in sorting the elements by weight/ value ratio.

2. I have used parallel merge sort with help of **#pragma omp parallel sections** construct.

3. We can clearly see a significant improvement in parallel sorting than sequential one.

4. One important thing to observe here is changing the knapsack capacity doesn't significantly affect the performance in parallel and sequential ones as parallel code doesn’t give any improvisation on capacity.

5. And Greedy Approach doesn’t give optimized result everytime.

**Complexity:**

**Sequential**

1. Merge Sort gives O(nlogn) complexity in all best, average and worst case.
2. Once we have sorted input, iterating over all items O(n) in all cases.

n : number of items

Overall time complexity: O(nlogn)

**Parallel**

1. Parallel Merge Sort gives O(logn^3) complexity in all best, average and worst case.
2. Once we have sorted input, iterating over all items O(n) in all cases.

n : number of items

Overall time complexity: O(n) + O(logn^3) 🡪 O(n)