Group Project Report

CS 3364

Sarthak Dudhani, Logan Ferris, Abhinav Raj Gupta, Medhawi Niroula, Jacob Wargo

---

Part 1. Hybrid Sort

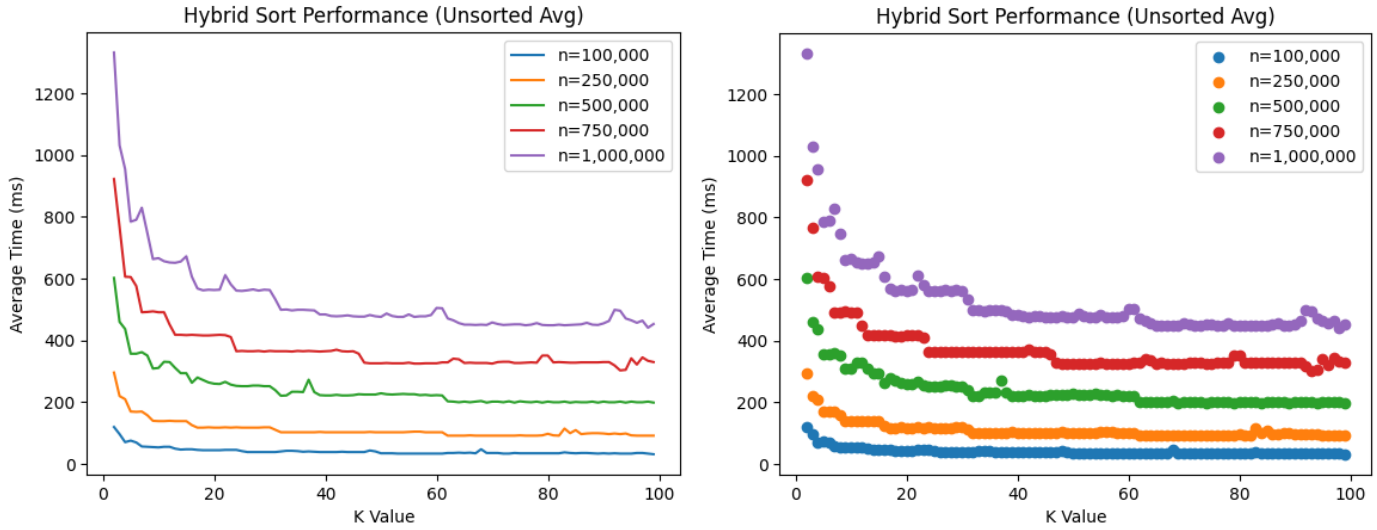
Deliverable 1.1

See the src folder in the submission file for the projects code:

- main.cpp
- mergeSort.cpp
- insertion_sort.cpp

Deliverable 1.2

A plot showing the average run time of your algorithm as a function of K, with a separate trace for at least 5 representative values of n. (20 points)



We met this expectation by finding the average runtime of the hybrid sort for k-values 2 through 99 for 5 different arrays of length n. The lengths of n we tested our hybrid sort algorithm on include 100,000, 250,000, 500,000, 750,000, and 1 million. Testing on these relatively large values of n gives us a good comparison when plotting each test on a graph, you can very easily see similarities between lines at certain k values, and you can clearly see outliers for each line.

Our program calculates the average by computing the runtime of each k-value 5 times on an array of length n and averaging out to find the average runtime. This did make testing take longer, but it gives accurate test results. See the plot above...

Each of the plot lines follows a similar trajectory. At first when the value of k is small, the average runtime is at its highest. Following the line as the value of k is increased, the average runtime decreases (until a certain point).

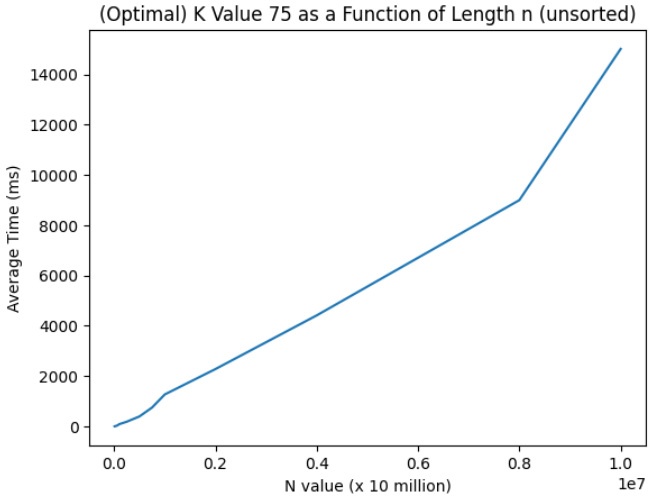
It is interesting to note that the larger the dataset is either the more outliers appear or the greater

the variance between the outlier and the average line is. Notice on the yellow line (250k) there are less random peaks appearing than compared to the purple line (1M) and the variance is greater. The same follows when comparing the blue line to the yellow line (100k vs 250k).

From the two plots, the optimal value of K seems to lie in the range of 60 to 80. I chose 75 as the optimal value because on each line plotted there are no outliers at that point, and the time has stabilized in a global valley.

Deliverable 1.3

A plot showing the optimal value of K as a function of array length n. Explain why you think the relationship between n and optimal K is the way that it is.(30 points)



Interestingly the line plot of the optimal value of K as a function of length n appears to be nlogn from 10 thousand to 1 million and becomes n (linear) from 1 million to 8 million. When testing the optimal value k on relatively small array sizes the results are not much different compared to if you were to use a different value of k from the graph, at lower values of n the hybrid sort is O(nlogn).

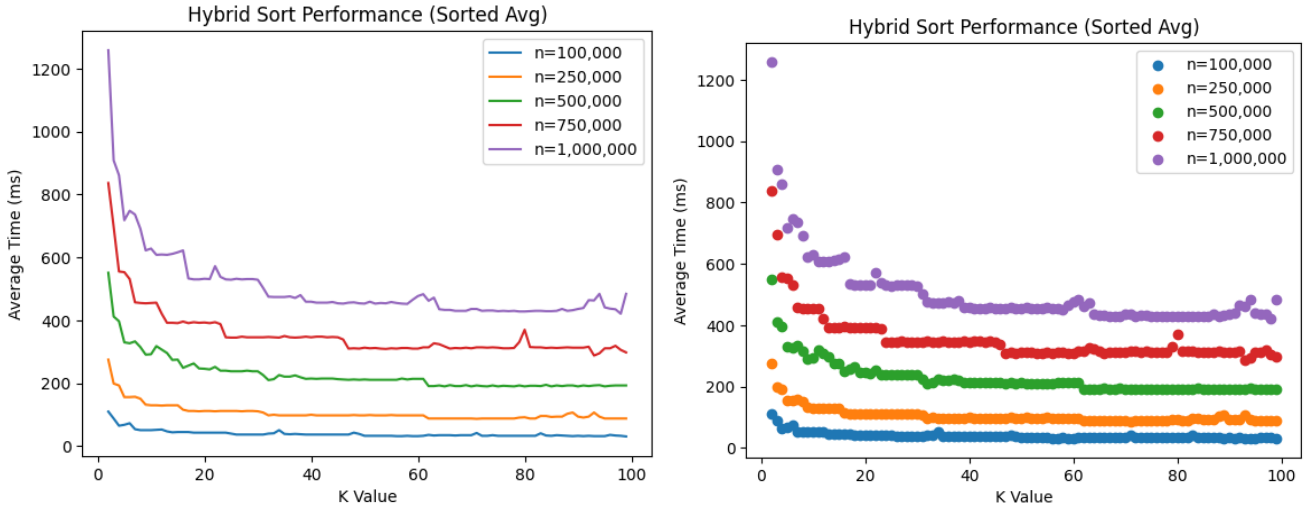
When the size of the array that you are testing on increases the time the algorithm takes to run also increases, but the efficiency of the hybrid sort increases. The optimal value of k and this graph shows us that at the k value 75 when you double the size of the arrays (1 million <= n <= 8 million) the average runtime doubles. This tells us that hybrid sort runs in an efficient amount of time for large data sets. From the graph, at this point the hybrid sort is O(n).

Deliverable 1.4

Your observations and findings from Task 4. (30 points)

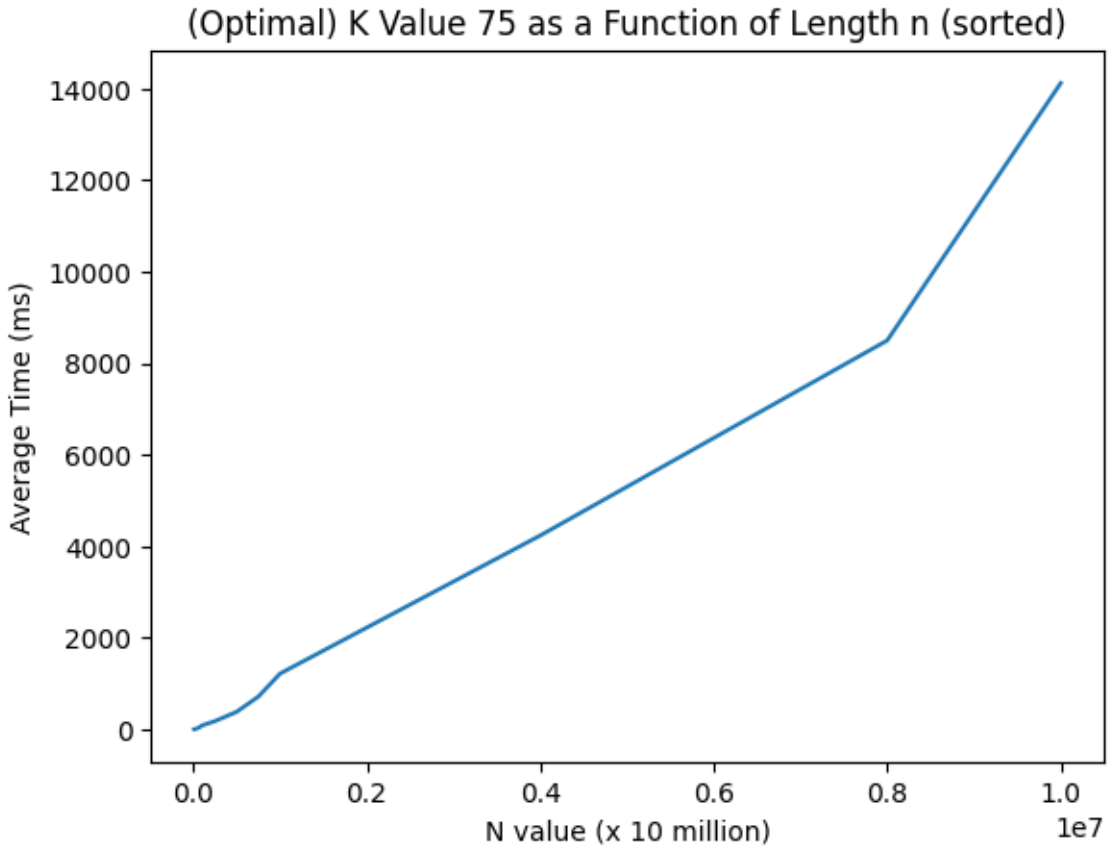
The graphs found for Deliverable 1.4 are similar to those found in the previous Deliverables 1.2 & 1.3. Looking at /deliverable_1.3_csv/sorted_csvs/values.csv and /deliverable_1.3_csv/unsorted_csvs/values.csv the runtime of hybrid sort on sorted arrays is slightly faster than sorting unsorted arrays. An average time difference of 118ms was found between sorted and unsorted arrays.

(1. 2) (sorted)



From the two plots, the optimal value of K seems to lie in the range of 60 to 80. Like the unsorted version, a K value of 75 seems to be the optimal value, the graph shows that at the value 75 no outliers exist on any of the 5 graphs and the average runtime has stabilized in a global valley. I chose the K value based on where the flattest and lowest part of all the lines are similar.

(1.3) (sorted)



The graph of the sorted array tests is like the graph of the unsorted array tests. Similarly, the line plot of the optimal value of K as a function of length n appears to be nlogn from 10 thousand to 1 million and becomes n (linear) from 1 million to 8 million. When testing the optimal value k on relatively small array sizes the results are not much different compared to if you were to use a different value of k from the graph, at lower values of n the hybrid sort is O(nlogn). The two graphs

sorted and unsorted have an interesting relationship because they are nearly identical. From this we conclude that when sorting an already sorted array, it will take a similar amount of time to sort an unsorted array using the same algorithm.

Part 2. Knapsack Problem

Deliverable 2.1

Sub Problem:

- Maximizing Enjoyment Points without Exceeding Luggage Weight Limit

The objective is to select a combination of experiences from Planet Xanadu in such a way that the total weight of the equipment required for these experiences does not exceed the luggage weight limit of 20 units, while maximizing the total enjoyment points that the tourists will receive.

Key Points:

Luggage Weight Limit:

- The maximum allowable weight that can be carried is 20 units.

Experiences and Equipment:

- Each experience has a specific weight and corresponding enjoyment points. The goal is to choose the most valuable combination of experiences within the weight limit.

Sub Problems Definition:

Let's consider each experience one by one. We have to determine whether including an experience maximizes enjoyment points while keeping the total weight within the limit.

The sub problem can be defined in terms of including or excluding each experience. First of all when including the experience, we have to add its weight to the total luggage weight and add its enjoyment points to the total enjoyment points, ensuring the total weight does not exceed 20 units. When we exclude an experience, we have to consider the next experience without adding its weight and enjoyment points to the total.

Mathematical Representation:

Let W be the luggage weight limit (20 units) and let each experience i have a weight wi and enjoyment point vi.

For each experience i and for each weight j from 0 to W, define the following point:

Value (Enjoyment Points) Array: V[i][j] represents the maximum enjoyment points obtainable using the first i experiences within weight limit j.

The Sub Problem can also be stated as:

- If we include experience i: V[i][j] = V[i-1][j - wi] + vi
- If we exclude experience i: V[i][j] = V[i-1][j]

The goal is to maximize V[n][W], where n is the number of experiences.
This formulation can be solved using dynamic programming, considering the recursive relationships outlined.

Deliverable 2.2

If the weight of the current item i is greater than the current capacity j, it cannot be included in the knapsack. Hence, the maximum value is the same as without this item. Also, If the weight of the current item i is less than or equal to the current capacity j, then we have two choices: either include the item or exclude it. We take the maximum value of these two choices. Therefore, the recurrence relation would be:

$$dp[i]\left[j\right] = \begin{cases} \mathrm{dp[i-1][j]} & \text{if} \quad \mathrm{weights[i-1} > \mathrm{j} \\ \max\left(\mathrm{dp[i-1][j]}, \mathrm{values[i-1]} + \mathrm{dp[i-1][j-weights[i-1]}\right] & \text{if} \quad \mathrm{weights[i-1]} \leq \mathrm{j} \end{cases}$$

Deliverable 2.3

See the src folder in the submission file for the projects code:

- Knapsack.py

Deliverable 2.4

According to the dynamic programming table in knapsack.py, experiences 3, 4, and 5 provide the highest value with a total of 6500 enjoyment points.

First of all, experiences 3, 4, and 5 both provide the highest experience values of the group with the lowest weight under the threshold. This means that they provide the highest value per unit of weight, with experience 5 providing 750 enjoyment points per unit of weight, experience 4 with 360 points per unit of weight, and 3 with 283.33 points per unit of weight. Meanwhile, experience 2 only provides 228.57 points per unit of weight, and experience 3 provides 187.5 points per unit of weight

Experience ID	Weight	Value	Points per Weight
1	8	1500	187.5
2	7	1600	228.57
3	6	1700	283.33
4	5	1800	360
5	4	3000	750