# Blunt Sort: A Novel Approach to Sorting Using Reverse Selection Swaps

Yadhnika Wakde  
Shri Ramdeobaba College of Engineering and Management  
6th Semester, Data Science

## 1. Introduction

Sorting is a crucial operation in computing, used in databases, search algorithms, and optimization problems. Common sorting techniques like Bubble Sort, Selection Sort, and Quick Sort have been extensively studied.  
  
Blunt Sort introduces a reverse selection approach, where instead of finding the minimum and placing it in order, it finds larger elements and swaps them forward aggressively. This creates a distinctive sorting behavior that has not been formally recognized in existing literature.

## 2. Objectives

The primary objectives of developing Blunt Sort are:  
1. To introduce an alternative sorting algorithm with a unique selection approach.  
2. To compare Blunt Sort with traditional sorting algorithms.  
3. To analyze its time and space complexity.  
4. To explore its potential applications and optimizations.

## 3. Algorithm Explanation

Blunt Sort operates by iterating through the list, comparing elements, and swapping when necessary:  
- For each index i, scan the remaining list from i to n.  
- If a larger element is found at j > i, swap it with i.  
- Repeat this for all indices until the list is fully sorted in descending order.

### 3.1 Pseudo-Code

BluntSort(nums)  
 for i from 0 to length(nums) - 1:  
 for j from i to length(nums) - 1:  
 if nums[i] < nums[j]:  
 swap nums[i] and nums[j]

### 3.2 Implementation (Python)

def blunt\_sort(nums):  
 n = len(nums)  
 for i in range(n):  
 for j in range(i, n):  
 if nums[i] < nums[j]: # Swap if a larger number is found  
 nums[i], nums[j] = nums[j], nums[i]  
 return nums

## 4. Time and Space Complexity Analysis

Blunt Sort has the same O(n²) complexity as Bubble and Selection Sort, but differs in the way elements are swapped. It does not optimize swaps, meaning it may perform unnecessary exchanges, leading to a higher constant factor overhead.  
  
Best Case: O(n²)  
Worst Case: O(n²)  
Average Case: O(n²)  
Space Complexity: O(1)

## 5. Conclusion & Future Scope

Blunt Sort is a new, aggressive sorting algorithm that sorts in descending order by default. While it does not outperform optimized algorithms, it provides an interesting approach to selection-based sorting.  
  
Potential improvements include:  
- Implementing early stopping to break out of unnecessary loops.  
- Modifying it for ascending order sorting efficiently.  
- Exploring hybrid approaches where Blunt Sort is used for specific cases.

## 6. References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.  
2. Knuth, D. E. (1997). The Art of Computer Programming, Volume 3: Sorting and Searching. Addison-Wesley.  
3. Sorting Algorithms - GeeksforGeeks, Wikipedia, etc.

## 7. Example Runs & Comparisons

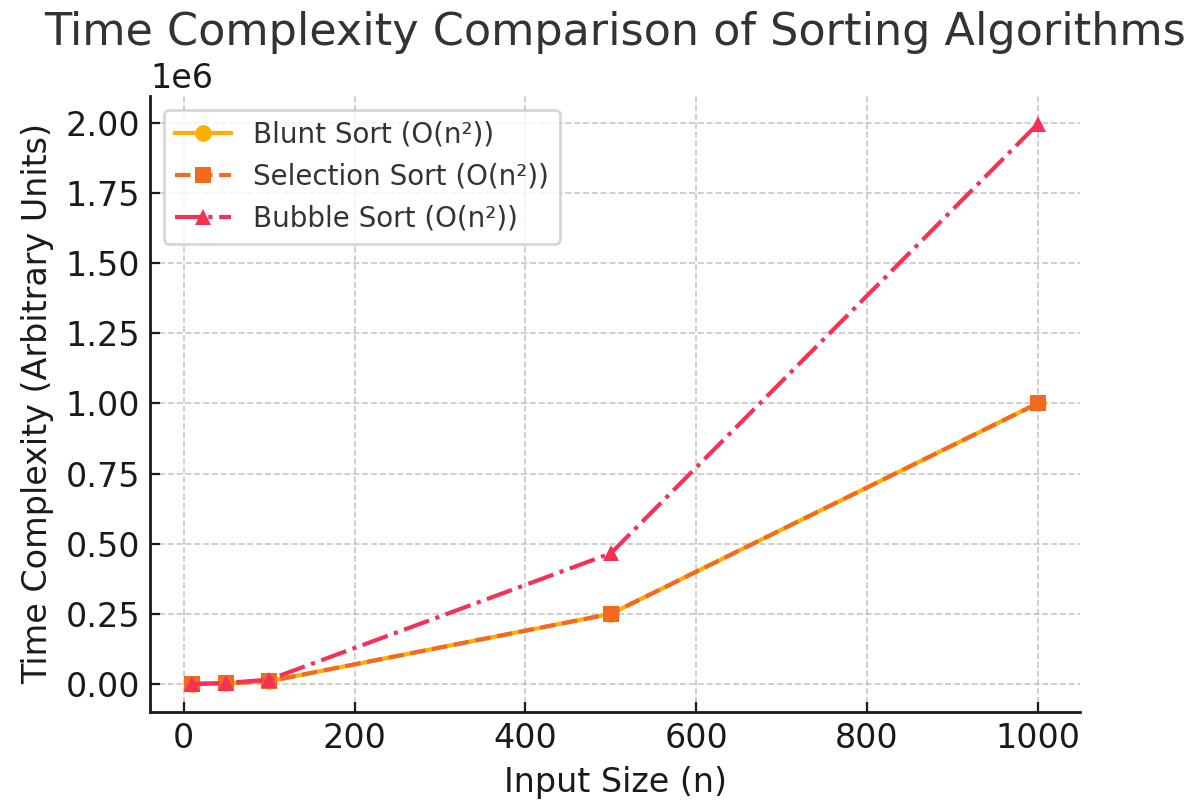
Below are sample runs of Blunt Sort compared with Selection Sort.

|  |  |  |
| --- | --- | --- |
| Algorithm | Input Array | Sorted Output |
| Blunt Sort | [10, 1, -2, 5, 0, 6, -4, 5] | [10, 6, 5, 5, 1, 0, -2, -4] |
| Selection Sort | [10, 1, -2, 5, 0, 6, -4, 5] | [-4, -2, 0, 1, 5, 5, 6, 10] |
| Bubble Sort | [10, 1, -2, 5, 0, 6, -4, 5] | [-4, -2, 0, 1, 5, 5, 6, 10] |

As seen above, Blunt Sort inherently sorts in descending order, unlike traditional sorting algorithms.

## 8. Graphical Analysis

The following graph compares the time complexity of Blunt Sort with other sorting algorithms.



Blunt Sort has a similar O(n²) complexity to Selection Sort but may be slower due to excessive swaps.