Алгоритми Сортування

Виконав: Труш Остап

Опис Постановки Задачі та Експерименту

Порівняти ефективність роботи чотирьох алгоритмів: Selection sort, Insertion sort, Merge sort, Shellsort для наступних чотирьох експериментів:

1. випадковим чином згенерований масив (згенерувати 5 експериментів та записати середнє значення порівнянь та затраченого часу для кожного з розмірів масивів та алгоритму) ;
2. значення масиву відсортовані у порядку зростання;
3. значення масиву відсортовані у порядку зменшення;
4. масив містить лише елементи з множини {1, 2, 3} - тобто в масиві багато елементів, які повторюються; згенерувати 3 експерименти (шляхом перестановки значень масивів) та записати середнє значення.

Кожен із експериментів проводиться на масивах розміром від 2^7 до 2^15 зі збільшенням розміру масиву на 2 на кожному кроці. Тобто на масивах розмірами 2^7, 2^8, 2^9, 2^10, 2^11, 2^12, 2^13, 2^14, 2^15.

Також кожен з 4 алгоритмів сортування перевіряється на копіях одного й того самого масиву всі чотири експерименти та для кожного з 9 розмірів масиву.

У 4 експериментах потрібно обчислити середнє значення кількості порівнянь та часу виконання алгоритму для кожного алгоритму сортування та відповідного розміру.

Будуть наведені графіки зростання кількості порівнянь та часу виконання для кожного алгоритму в залежності від розміру вхідного масиву(8 графіків, по 2 на кожен з 4 експериментів). Графіки - логарифмічні, вісь x - степінь розміру масиву, вісь y - кількість порівнянь або витрачений час на обчислення, представлені як 10^X.

Специфікації Комп'ютера на якому проводився експеримент

- Операційна Система: **Arch Linux**
- Ядро|Кількість пакетів: **5.14.8-arch1-1 | 1113 Pacman Пакетів.**
- Процессор: **Intel® Core ™ i7-10510U (8M Cache, 4.9 GHz); 4 ядра, 8 потоків.**
- Оперативна пам'ять: **16 ГБ DDR4**
- Постійна пам'ять: **SSD 512 ГБ**

Програмний Код

Весь програмний код для алгоритмів сортування, аналізу даних та створення графіків був написаний на мові програмування ***Python*** та бібліотеці ***Matplotlib***(Графіки).

Усе це, та ще зображення графіків доступні на Github репозиторії за посиланням:

https://github.com/Adeon18/AlgorithmEfficiencyComparison

Директорія ***src*** містить весь програмний код, директорія ***data*** містить два json файли з даними:

- raw_test_results.json - містить дані всіх експериментів без обчислення середніх значень. В списках random_x та інших містяться списки списків часу та кількості порівнянь для кожного розміру масивів.
- final_results.json - містить вже оброблені дані з усіма обчисленими середніми значеннями.

!Всі програмні файли необхідно запускати з src директорії!

Містять просту документацію та опис застосування.

Зображення графіків містяться в директорії ***img***.

Аналіз Отриманих Даних

Базові Дані про Алгоритми

Надано `Best` або `Average` складність в залежності від різниці одне з одним

Shell Sort:

- Average: *O(nlogn)*
- Worst: *O(n^2)*
- Space Complexity: *O(1)*
- Stability: *No*

Merge Sort:

- Average: *O(nlogn)*
- Worst: *O(nlogn)*
- Space Complexity: *O(n)*
- Stability: *Yes*

Insertion Sort:

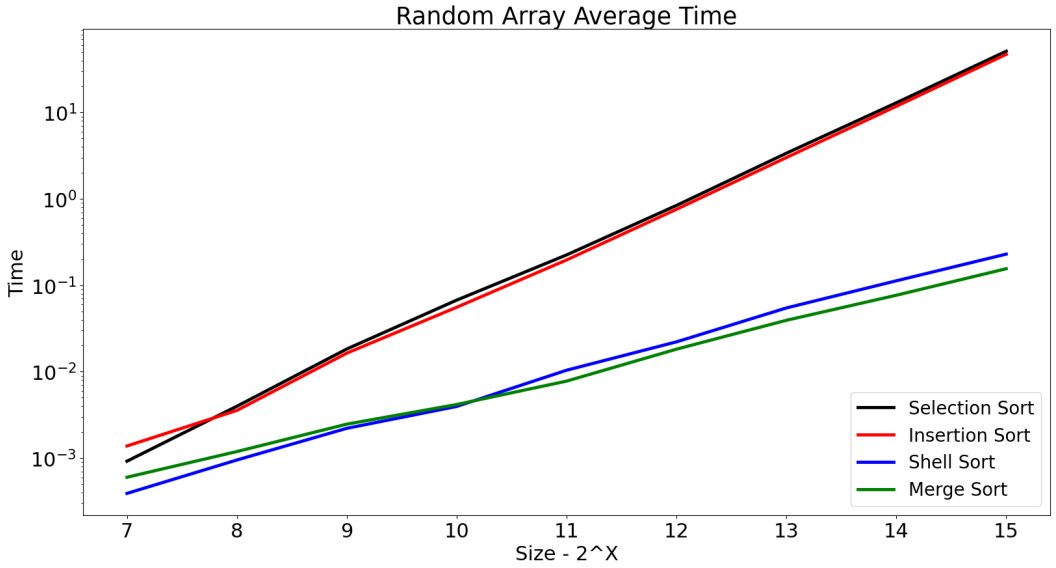
- Best: *O(n)*
- Worst: *O(n^2)*
- Space Complexity: *O(1)*
- Stability: *Yes*

Selection Sort:

- Best: *O(n^2)*
- Worst: *O(n^2)*
- Space Complexity: *O(1)*
- Stability: *No*

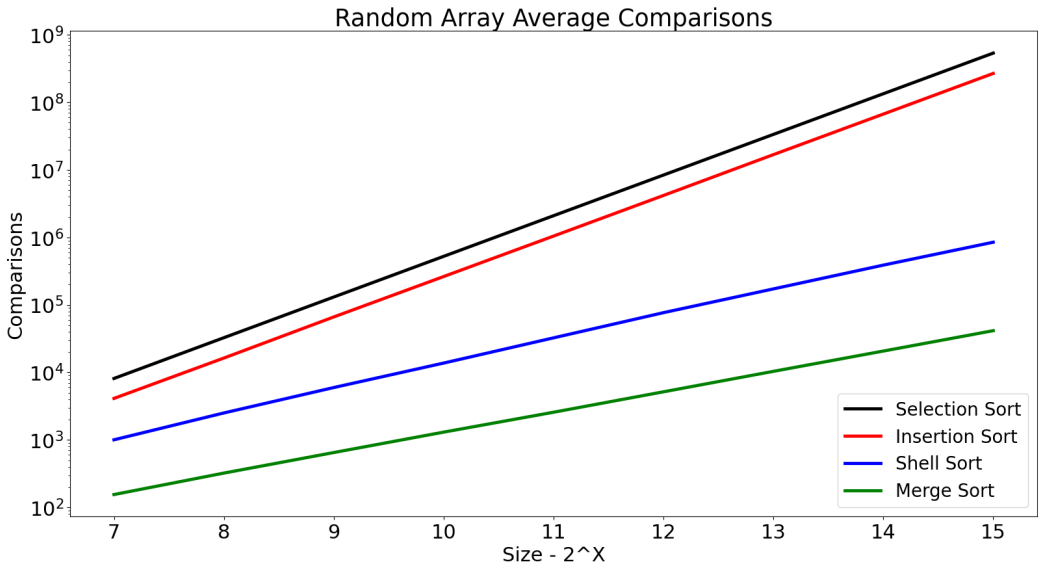
Масив з Рандомними Елементами

Було проведено 5 експериментів для масивів розмірами 2^7, 2^8, 2^9, 2^10, 2^11, 2^12, 2^13, 2^14, 2^15. Масиви містили рандомну кількість цілих чисел від 1 до 1000.



По часу ми бачимо що Insertion Sort і Selection Sort зайняли найбільше часу через їхню складність O(n^2) для більшості, а то і всіх(Selection Sort) масивів. Час в середньому сягав біля 100 секунд для найдовшого масиву.

Shell Sort, будучи швидшою версією Insertion Sort справився до секунди для найбільшого масиву, через середню складність O(nlogn). Так само Merge Sort показав всю силу рекурсивного підходу “розділяй та володарюй” та через складність O(nlogn) впорався до секунди. Різниця лише в тому, що йому для цього потрібні додаткові ресурси.

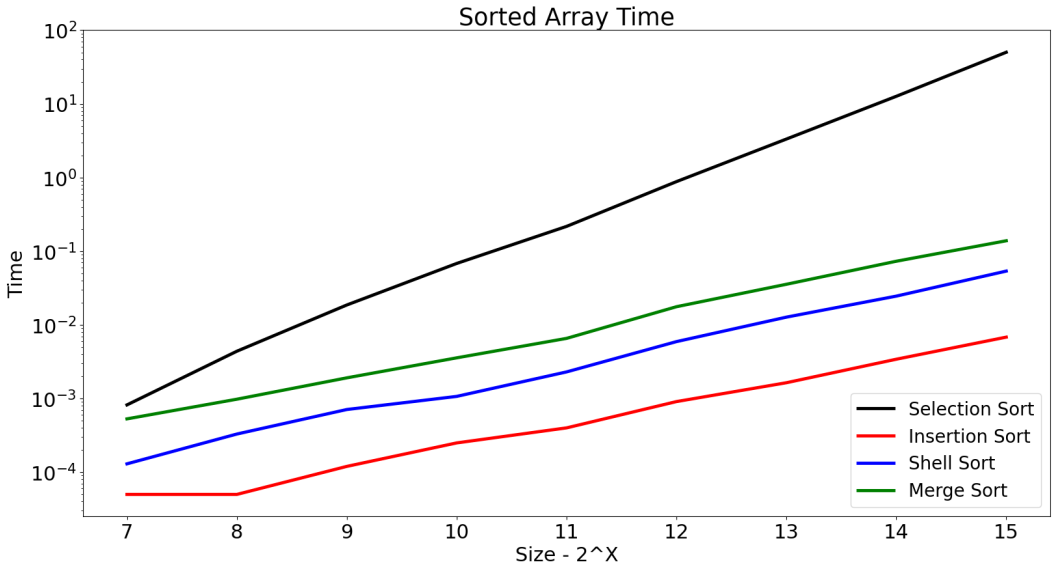


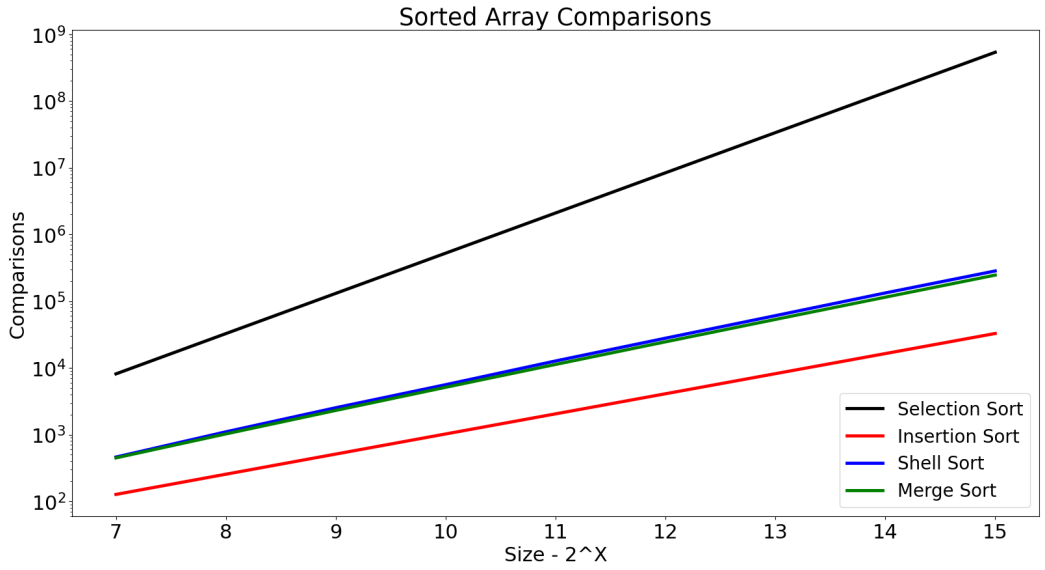
Щодо кількості порівнянь, все майже аналогічно з попереднім, лише `Insertion Sort` показує що при більш сприятливій розстановці елементів в масиві, він може демонструвати дещо меншу кількість порівнянь ніж `Selection Sort`.

`Merge Sort` має в рази менше порівнянь ніж `Shell Sort`, через використання додаткових ресурсів та рекурсивних викликів, до чого `Shell Sort` не може зрівнятися через `Insertion Sort` основу.

Масив з Посортованими Елементами

Як і в минулому прикладі, все так само тільки масив посортований, елементи становлять від [1 до довжини масиву] та експеримент лише один.





Тут `Insertion Sort` - найшвидший алгоритм і робить найменше порівнянь, адже він працює за лінійний час для посортованих масивів - `O(n)`.

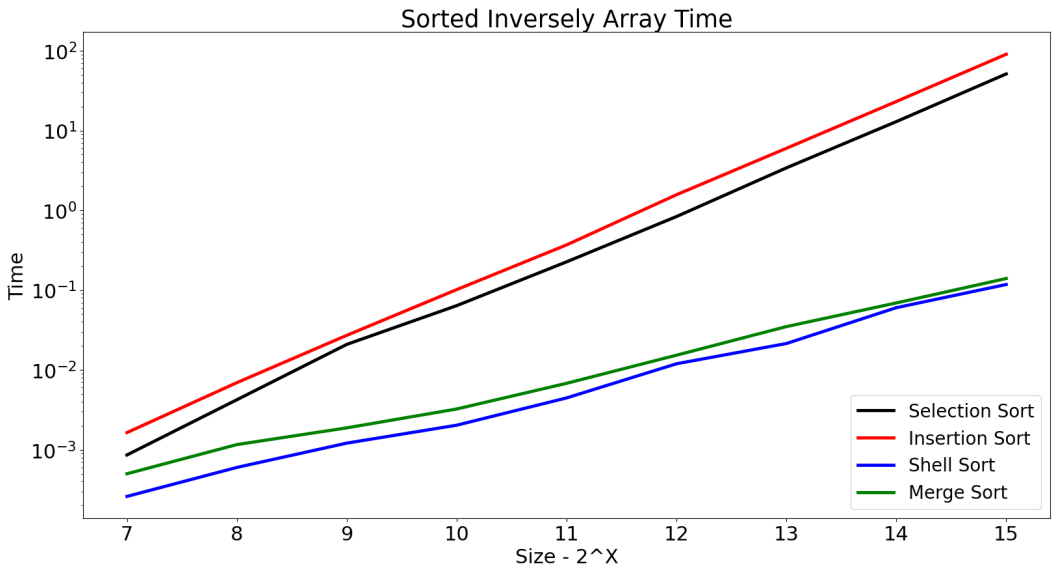
Shell Sort - будучи вдосконаленою імплементацією Insertion Sort, але для погано посортованих масивів, найшвидше може працювати за O(nlogn), адже використовує підхід “divide and conquer”.

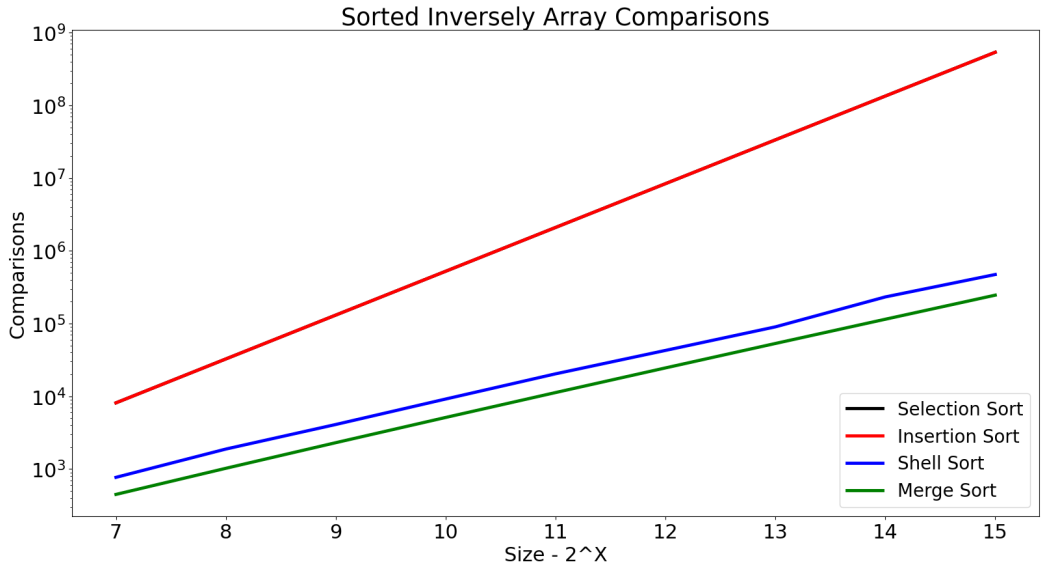
Merge Sort - Йде на рівні з Shell Sort по порівняннях та після нього по часу. Працює за O(nlogn). Merge Sort не має ніяких перевірок взятих від Insertion Sort, на відміну від Shell Sort, і використовує додаткові ресурси комп’ютера.

З Selection Sort все зрозуміло - O(n^2) завжди.

Масив з Посортованими елементами в іншу сторону

Все так само як і у минулому експерименті, лише масиви посортовані від найбільшого до найменшого.



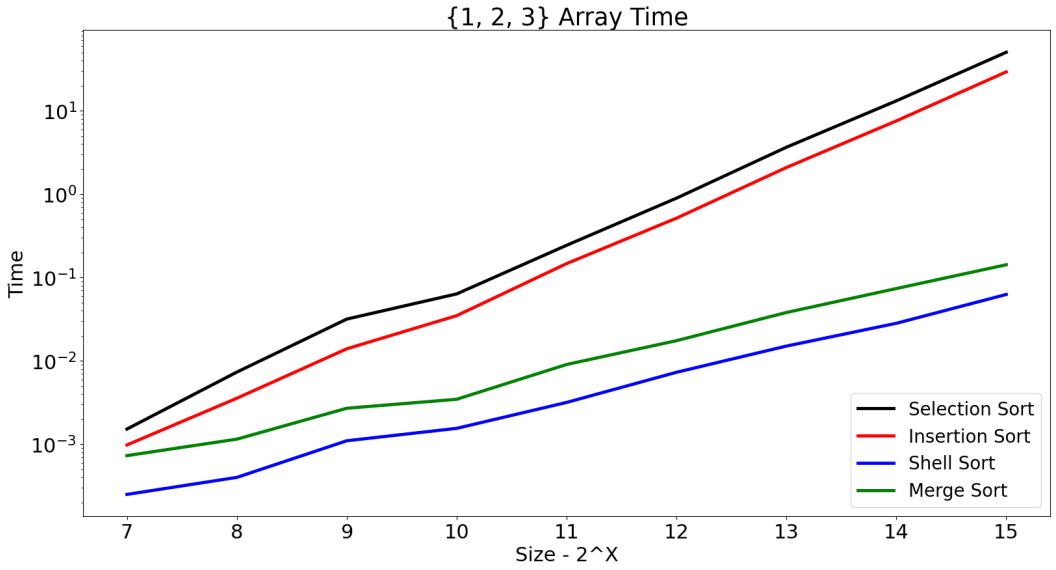


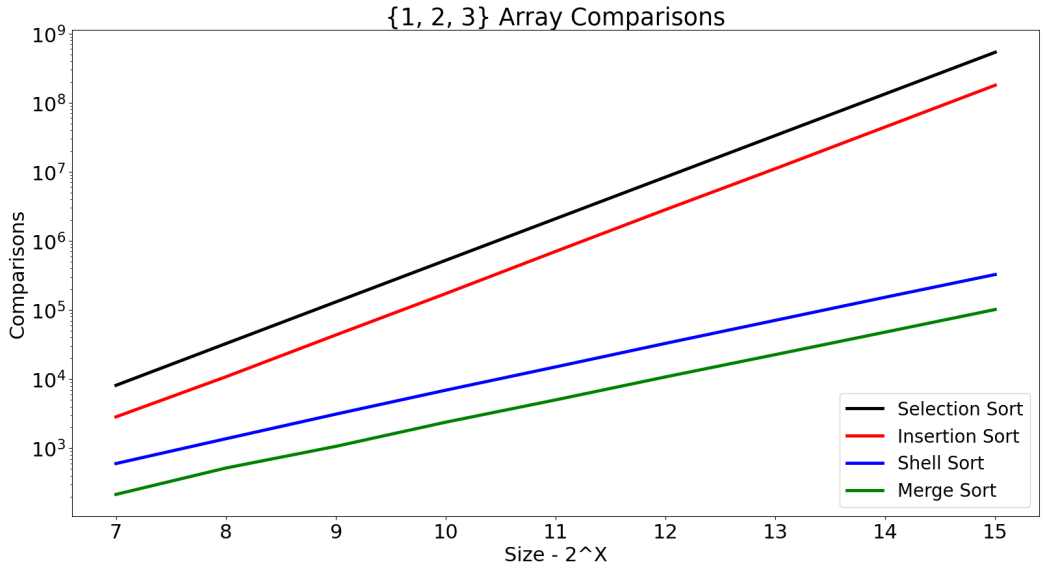
Це є найгірший випадок для `Insertion Sort` і він тут працює з точно такою самою складністю як `Selection Sort` працює завжди - `O(n^2)`.

Shell Sort і Merge Sort працюють краще, за O(nlogn), роблячи майже однакову кількість порівнянь та працюючи за близький до одинакового час. Тут в основному грає роль підхід “divide and conquer”, який є дуже ефективним для таких вхідних даних.

Масив з Елементами з Множини {1, 2, 3}

З Експерименти, елементи з множини {1, 2, 3} перемішані для кожного експерименту. Розміри масивів такі самі.





Загалом цей експеримент дуже схожий до першого, але тут по часу перемагає Shell Sort. Це стається через покрокове ділення масиву на

менші частини та відкидання більших елементів назад, а потім зменшення кроку. Оскільки в нас є лише один більший елемент тут - 3, він завжди буде переміщатися в кінець поточного підмасиву на кожному кроці ділення. Також хоч і Merge Sort робить менше порівнянь, різниця між порівняннями Shell Sort і Merge Sort значно скоротилася, якщо порівняти з першим прикладом.

Insertion Sort і Selection Sort тут працюють аналогічно з першим прикладом. Але Insertion Sort трохи збільшив відрив від Selection Sort в часі і кількості порівнянь через меншу різноманітність елементів.

Підсумки

Дослідивши роботу алгоритмів сортування Merge Sort, Selection Sort, Insertion Sort, Quick Sort, можна зробити висновки про кожен із алгоритмів і коли його застосовувати.

Insertion Sort - Загалом доволі повільний алгоритм, але неймовірно ефективний для посортованих або майже посортованих масивів, де демонструє лінійну складність(O(n)) і є найефективнішим з усіх досліджених. Найгірший випадок - масив, посортованих в порядку спадання - O(n^2)

Merge Sort - Ефективний для всіх масивів та завжди демонструє складність O(nlogn). Поступається Shell Sort та Insertion Sort в посортованих масивах та Shell Sort по часу у масиві з елементами з множини {1, 2, 3}. Найкраще використовувати у великих, не посортованих масивах з великою кількістю різних елементів та коли є доступні додаткові ресурси комп'ютера.

Shell Sort - В середньому випадку складність O(nlogn). Дуже ефективний для масивів з великою кількістю не посортованих елементів з маленької множини(Пр. {1, 2, 3}). Також не потребує додаткових ресурсів комп'ютера і може бути чудовою заміною Merge Sort якщо ресурси обмежені.

Selection Sort - Не варто використовувати, адже складність завжди O(n^2).