ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

------📖------

BÀI BÁO CÁO BÀI TẬP VỀ NHÀ

Giảng viên hướng dẫn: Phan Quốc Kỳ

Lớp: 24CTT3

1 Giới thiệu

Trong bài tập tuần 3, Set 2, nhóm chúng tôi đã thực hiện việc so sánh hiệu suất của các thuật toán sắp xếp bao gồm 11 thuật toán: Selection Sort, Insertion Sort, Bubble Sort, Shaker Sort, Shell Sort, Heap Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort và Flash Sort. Các thuật toán này được thực hiện trên các bộ dữ liệu với kích thước và dạng khác nhau, nhằm đo lường thời gian chạy và số phép so sánh của mỗi thuật toán.

2 Môtả các thuật toán

• Bubble Sort: Bubble Sort hoạt động bằng cách duyệt qua mảng và so sánh các cặp phần tử liền kề, hoán đổi chúng nếu chúng không theo đúng thứ tự. Quá trình này được lặp đi lặp lại cho đến khi không có phép hoán đổi nào được thực hiện, nghĩa là mảng đã được sắp xếp. Đây là thuật toán đơn giản, dễ hiểu nhưng có độ phức tạp thời gian là O(n2), làm cho nó không hiệu quả đối với các mảng lớn.

• Selection Sort: Selection Sort tìm kiếm phần tử nhỏ nhất trong mảng và hoán đổi nó với phần tử đầu tiên, sau đó tiếp tục tìm kiếm phần tử nhỏ nhất trong phần còn lại của mảng và hoán đổi với phần tử thứ hai, cứ như vậy cho đến khi mảng được sắp xếp. Thuật toán này có độ phức tạp thời gian là O(n2) và thực hiện số lượng hoán đổi ít, nhưng vẫn không hiệu quả với mảng lớn.

• Shell Sort: Shell Sort là một cải tiến của thuật toán Insertion Sort. Thay vì so sánh các phần tử liền kề, nó sử dụng một khoảng cách (gap) giữa các phần tử cần so sánh. Gap giảm dần cho đến khi trở thành 1, lúc đó thuật toán giống như Insertion Sort. Thuật toán này có thể có độ phức tạp O(n2) trong trường hợp xấu nhất, nhưng có hiệu suất tốt hơn Insertion Sort trong nhiều tình huống.

• Counting Sort: Counting Sort sử dụng một mảng phụ để đếm số lần xuất hiện của từng giá trị trong mảng đầu vào. Sau khi đếm, nó tái tạo lại mảng đã sắp xếp dựa trên số lần xuất hiện của các phần tử. Đây là thuật toán không sử dụng so sánh và có độ phức tạp O(n+k), trong đó n là số phần tử trong mảng và k là giá trị lớn nhất trong mảng. Counting Sort rất hiệu quả khi phạm vi giá trị trong mảng là nhỏ. 1

• Merge Sort: Merge Sort là thuật toán chia để trị. Nó chia mảng thành các phần nhỏ hơn cho đến khi mỗi phần chỉ có một phần tử. Sau đó, các phần này được kết hợp lại theo thứ tự sắp xếp. Merge Sort có độ phức tạp O(nlogn) trong mọi trường hợp và là một thuật toán ổn định, tuy nhiên, nó yêu cầu bộ nhớ phụ để lưu trữ các mảng con trong quá trình hợp nhất.

• Flash Sort: Flash Sort phân chia mảng thành các dải (buckets) dựa trên giá trị của các phần tử. Sau khi phân chia, thuật toán sử dụng một phương pháp sắp xếp nội bộ (thường là Insertion Sort) cho mỗi dải. Flash Sort có thể đạt hiệu suất rất tốt khi các phần tử trong mảng có phân phối đều. Thuật toán này có thể đạt được độ phức tạp O(n) trong các trường hợp lý tưởng, nhưng có thể kém hiệu quả nếu dữ liệu không phân phối đều.

• Insertion Sort: Insertion Sort hoạt động bằng cách lần lượt đưa các phần tử vào vị trí đúng trong mảng đã sắp xếp. Mỗi phần tử được đưa vào vị trí của nó trong mảng đã sắp xếp bằng cách dịch chuyển các phần tử lớn hơn về phía sau. Đây là thuật toán đơn giản và rất hiệu quả khi

mảng đã gần như được sắp xếp hoặc đối với các mảng nhỏ. Tuy nhiên, độ phức tạp trong trường hợp tồi tệ là O(n2).

• Shaker Sort: Shaker Sort là một biến thể của Bubble Sort, trong đó thuật toán duyệt mảng theo cả hai chiều: từ trái qua phải và từ phải qua trái. Điều này giúp giảm bớt số lượt duyệt trong một số trường hợp nhất định. Mặc dù có sự cải tiến so với Bubble Sort, Shaker Sort vẫn có độ phức tạp O(n2) và không phù hợp với mảng lớn.

• Heap Sort: Heap Sort sử dụng cấu trúc dữ liệu heap để sắp xếp mảng. Đầu tiên, nó xây dựng một heap từ mảng. Sau đó, phần tử lớn nhất (hoặc nhỏ nhất) được lấy ra và đưa vào đúng vị trí của nó trong mảng. Thuật toán có độ phức tạp O(nlogn) và không cần bộ nhớ phụ ngoài mảng gốc, nhưng không phải là thuật toán ổn định.

• Quick Sort: Quick Sort sử dụng phương pháp chia để trị, chọn một phần tử làm pivot, sau đó phân chia mảng thành hai phần sao cho các phần tử nhỏ hơn pivot nằm ở một bên và các phần tử lớn hơn pivot nằm ở bên kia. Thuật toán tiếp tục đệ quy sắp xếp các phần này. Quick Sort có độ phức tạp trung bình là O(nlogn), nhưng trong trường hợp xấu (khi pivot không được chọn tốt), độ phức tạp có thể lên tới O(n2).

• Radix Sort: Radix Sort là một thuật toán sắp xếp không sử dụng so sánh. Nó sắp xếp các phần tử dựa trên các chữ số của chúng từ thấp 2 đến cao. Thuật toán này rất hiệu quả với các dãy số nguyên, đặc biệt là khi số chữ số của các phần tử nhỏ. Độ phức tạp của Radix Sort là O(n · k), trong đó k là số chữ số tối đa của phần tử lớn nhất trong mảng.

3 Các thí nghiệm Trong bài tập này, các thí nghiệm được thực hiện theo các bước sau:

1. Tạo một mảng với thứ tự dữ liệu và kích thước nhất định.

2. Sắp xếp mảng bằng thuật toán đã chọn, đồng thời đo thời gian chạy và số phép so sánh.

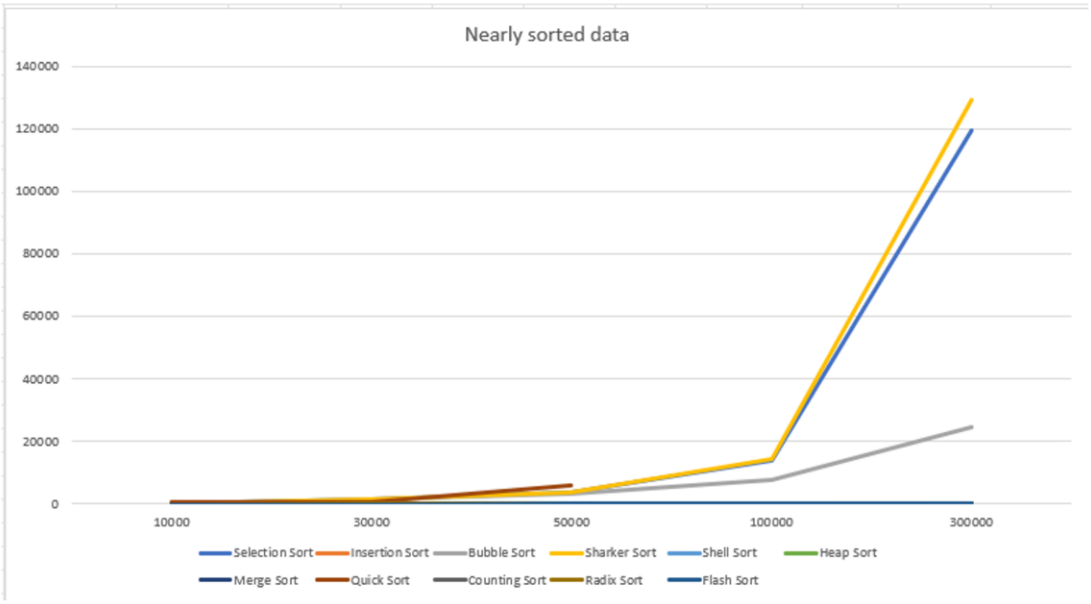
3. Ghi lại kết quả của mỗi thí nghiệm. Các thí nghiệm được thực hiện với các bộ dữ liệu theo các dạng khác nhau:

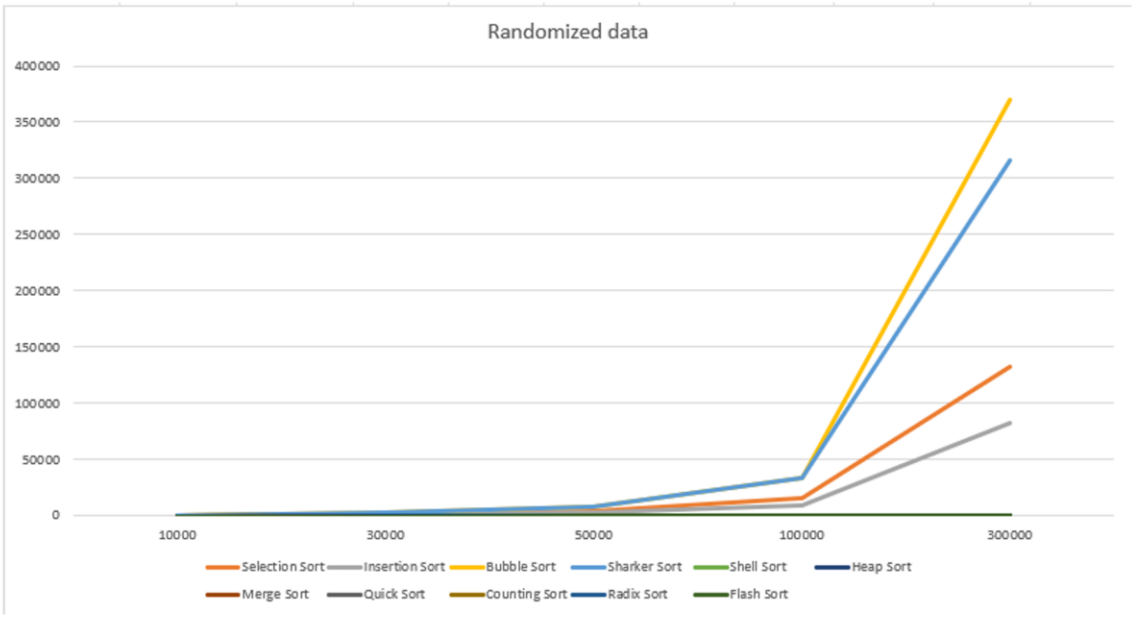
- Sorted data
- Nearly sorted data
- Reverse sorted data
- Randomized data

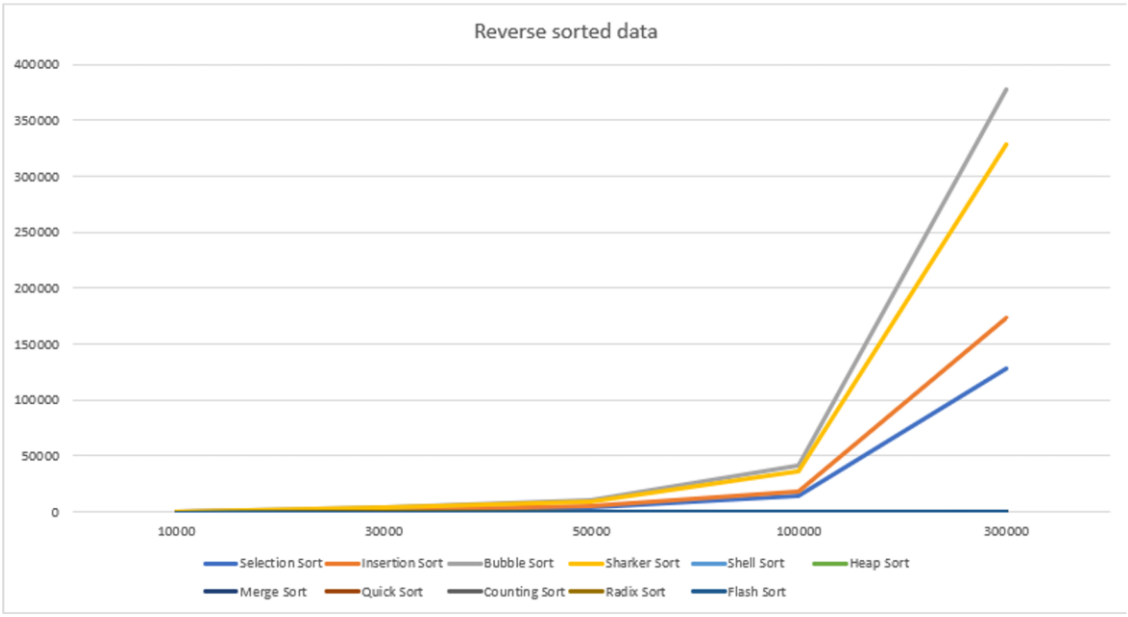
- Ngoài ra, các kích thước dữ liệu được sử dụng trong các thí nghiệm là: 10,000, 30,000, 50,000, 100,000 và 300,000 phần tử.

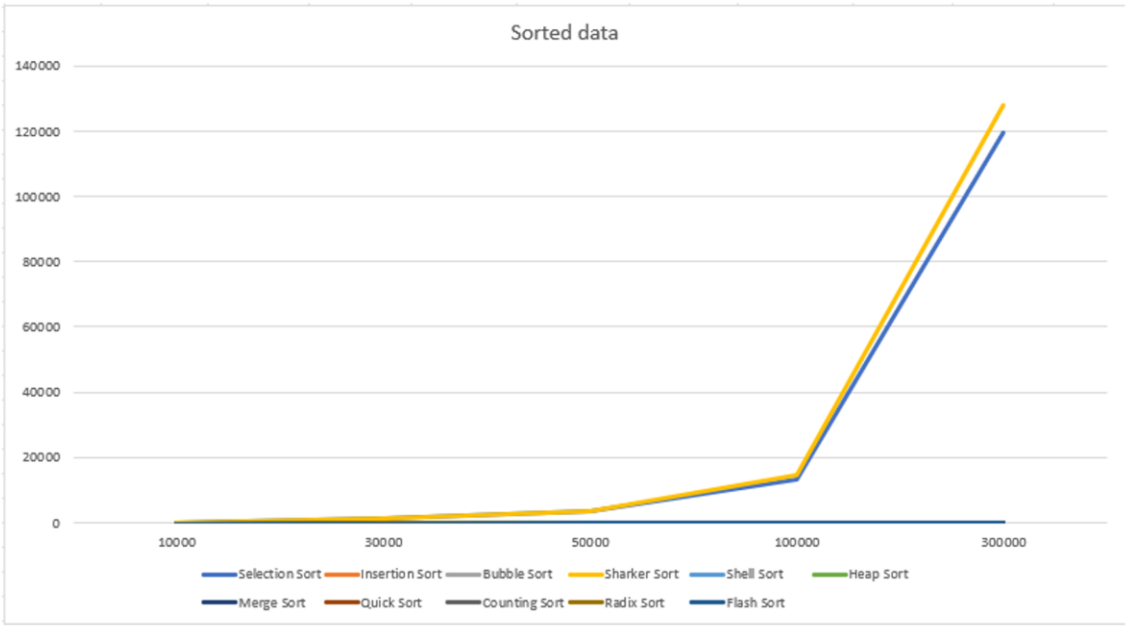
4 Kết quả

4.1 Biểu đồ thời gian chạy Dưới đây là các biểu đồ thể hiện thời gian chạy của các thuật toán sắp xếp trên các dữ liệu đã sắp xếp, gần như đã sắp xếp, sắp xếp ngược và dữ liệu ngẫu nhiên.

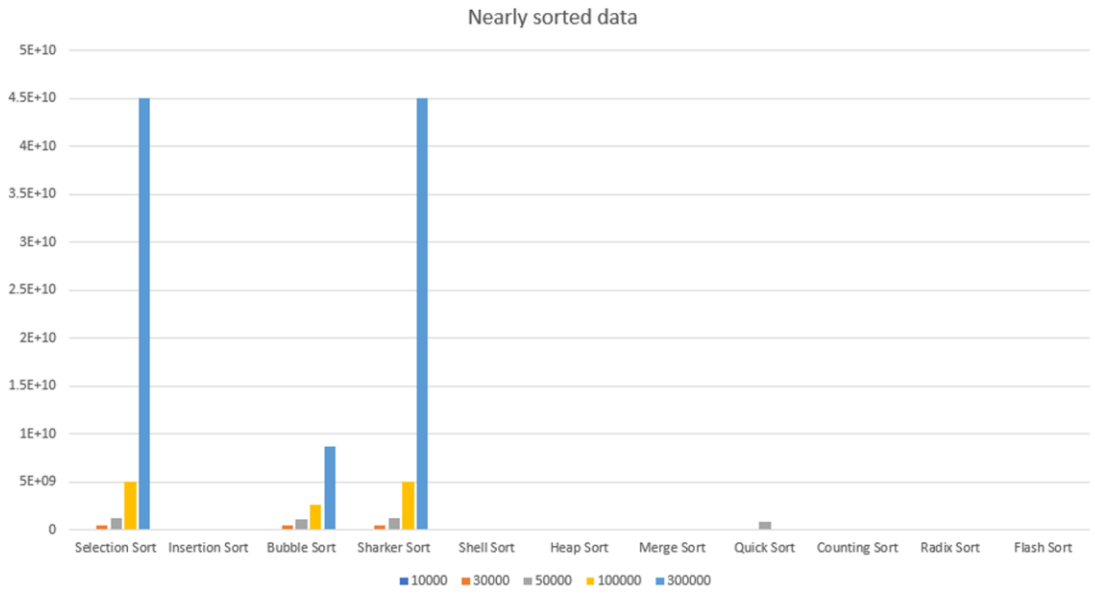

Nearly sorted data
140000
120000
100000
80000
60000
40000
20000
0
10000
30000
50000
100000
300000
Selection Sort
Insertion Sort
Bubble Sort
Sharker Sort
Shell Sort
Heap Sort
Merge Sort
Quick Sort
Counting Sort
Radix Sort
Flash Sort

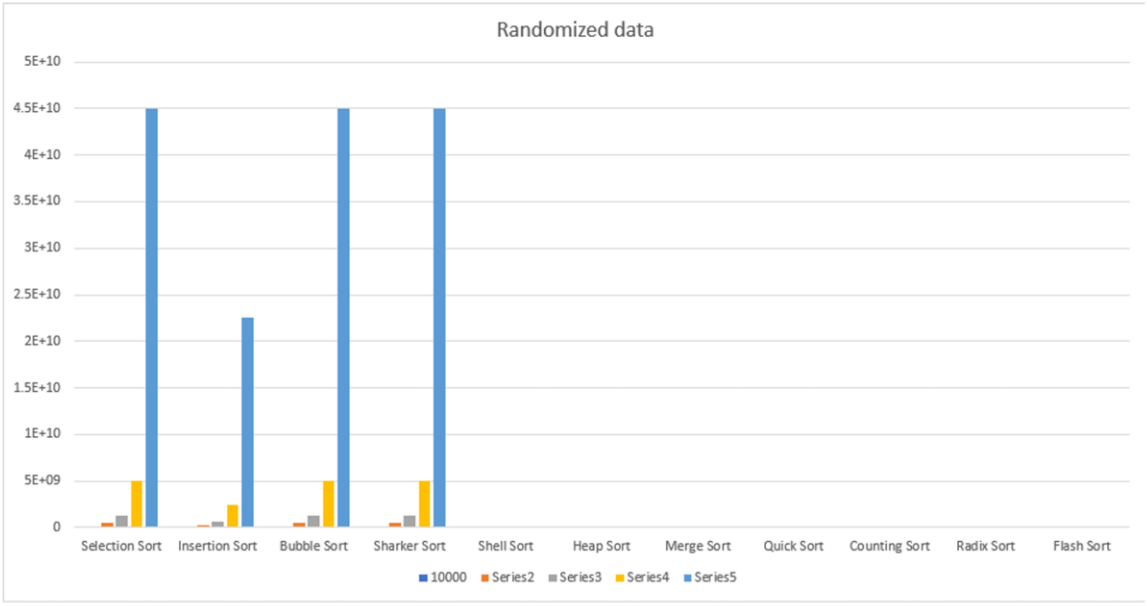

Randomized data
400000
350000
300000
250000
200000
150000
100000
50000
0
10000
30000
50000
100000
300000
Selection Sort
Insertion Sort
Bubble Sort
Sharker Sort
Shell Sort
Heap Sort
Merge Sort
Quick Sort
Counting Sort
Radix Sort
Flash Sort

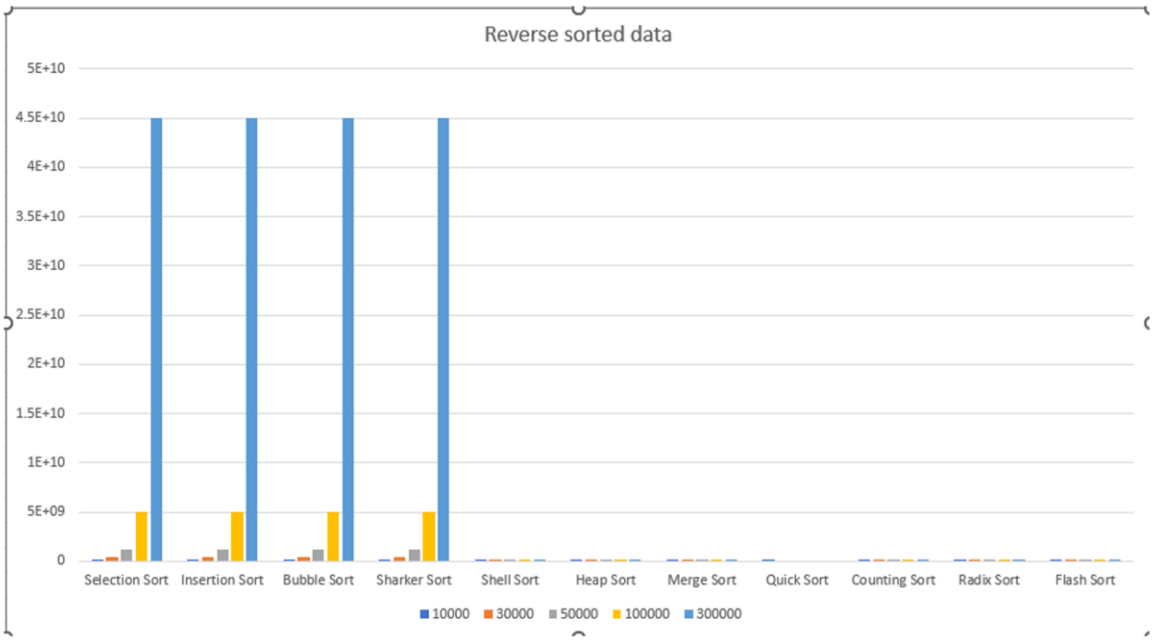


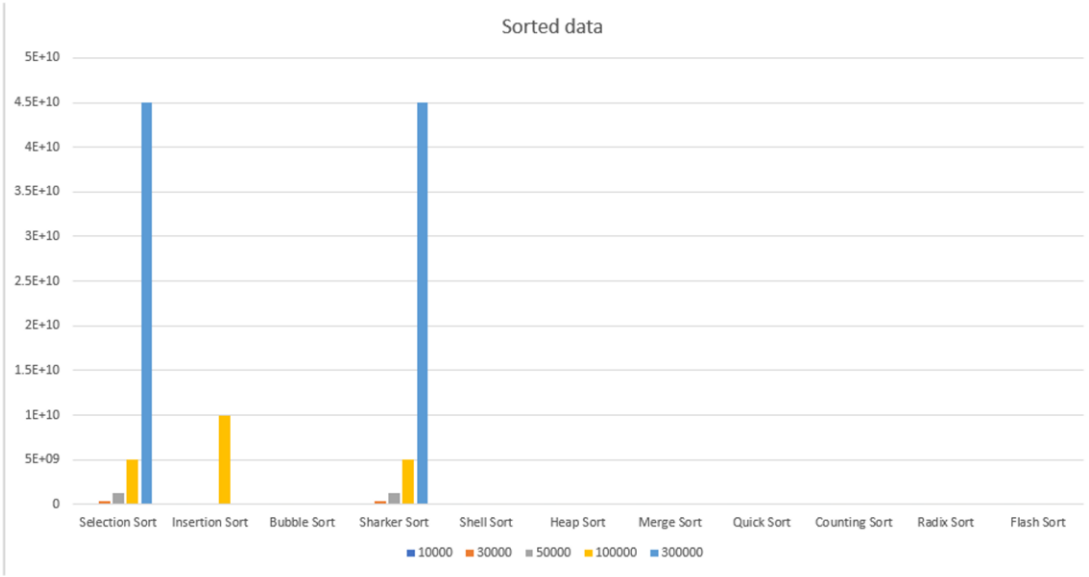


4.2 Biểu đồ số phép so sánh Dưới đây là các biểu đồ thể hiện số phép so sánh của các thuật toán sắp xếp trên các dữ liệu đã sắp xếp, gần như đã sắp xếp, sắp xếp ngược và dữ liệu ngẫu nhiên.


Nearly sorted data
5E+10
4.5E+10
4E+10
3.5E+10
3E+10
2.5E+10
2E+10
1.5E+10
1E+10
5E+09
0
Selection Sort
Insertion Sort
Bubble Sort
Sharker Sort
Shell Sort
Heap Sort
Merge Sort
Quick Sort
Counting Sort
Radix Sort
Flash Sort
10000
30000
50000
100000
300000


Randomized data
5E+10
4.5E+10
4E+10
3.5E+10
3E+10
2.5E+10
2E+10
1.5E+10
1E+10
5E+09
0
Selection Sort
Insertion Sort
Bubble Sort
Sharker Sort
Shell Sort
Heap Sort
Merge Sort
Quick Sort
Counting Sort
Radix Sort
Flash Sort
10000
Series2
Series3
Series4
Series5





4.3 Input & Output:

1. selection-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a selection-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

2. bubble-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a bubble-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

3. merge-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a merge-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

4. shell-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a shell-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

5. counting-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a counting-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

6. flash-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a flash-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

7. insertion-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a insertion-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

8. shaker-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a shaker-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

9. heap-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a heap-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

10. quick-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a quick-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

11. radix-sort

Example command: <đường dẫn thư mục>\main.exe -a <ten thuat toan> -i <file input> -o <file output>.

VD: D:\DSA\exercise-dsa\Week3\main.exe -a radix-sort -i input.txt -o output.txt

Input file: 2 3 4 5 6 1 7 8

Output file: 1 2 3 4 5 6 7 8

4.4 GitHub:

1. bubble-sort:

```cpp
//Thuật toán bubble sort
void bubble_sort(vector<int>& a) {
    int n = a.size();
    bool swapped;
    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1]) {
                swap(a[j], a[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}
```

2. selection-sort:

```cpp
//Thuật toán selection sort
void selection_sort(vector<int>& a) {
    int n = a.size();
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++) {
            if (a[j] < a[min]) {
                min = j;
            }
        }
        if (min != i) {
            swap(a[i], a[min]);
        }
    }
}
```

3. shell-sort:

```cpp
//Thuật toán shell sort
void shell_sort(vector<int>& a) {
    int n = a.size();
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = a[i];
            int j = i;
            while (j >= gap && a[j - gap] > temp) {
                a[j] = a[j - gap];
                j -= gap;
            }
            a[j] = temp;
        }
    }
}
```

4. counting-sort:

```cpp
//Thuật toán counting sort
void counting_sort(vector<int>& a) {
    if (a.empty()) return;
    int max_a = *max_element(a.begin(), a.end());
    vector<int> count(max_a + 1, 0);

    for (int num : a) {
        count[num]++;
    }

    int index = 0;
    for (int i = 0; i <= max_a; i++) {
        while (count[i] > 0) {
            a[index++] = i;
            count[i]--;
        }
    }
}
```

5. merge-sort:

```cpp
//Thuật toán merge sort
void merge(vector<int>& a, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    vector<int> l(n1), r(n2);

    for (int i = 0; i < n1; i++)
        l[i] = a[left + i];
    for (int i = 0; i < n2; i++)
        r[i] = a[mid + 1 + i];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (l[i] <= r[j]) {
            a[k] = l[i++];
        } else {
            a[k] = r[j++];
        }
        k++;
    }

    while (i < n1) a[k++] = l[i++];
    while (j < n2) a[k++] = r[j++];
}

void merge_sort(vector<int>& a, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        merge_sort(a, left, mid);
        merge_sort(a, mid + 1, right);
        merge(a, left, mid, right);
    }
}
```

6. flash-sort:

```cpp
//Thuật toán flash sort
void flash_sort(vector<int>& a) {
    int n = a.size();

    if (n <= 1) return;

    int min_arr = a[0], max_i = 0;
    for (int i = 1; i < n; i++) {
        if (a[i] < min_arr) min_arr = a[i];
        if (a[i] > a[max_i]) max_i = i;
    }
    if (a[max_i] == min_arr) return;

    int m = int(0.45 * n);
    vector<int> L(m, 0);
    double scale = (double)(m - 1) / (a[max_i] - min_arr);

    for (int i = 0; i < n; i++) {
        int index = scale * (a[i] - min_arr);
        L[index]++;
    }

    for (int i = 1; i < m; i++) {
        L[i] += L[i - 1];
    }

    int i = 0, count = 0;
    while (count < n) {
        int index = scale * (a[i] - min_arr);
        while (i >= L[index]) {
            index = scale * (a[++i] - min_arr);
        }

        int temp = a[i];
        while (i != L[index]) {
            index = scale * (temp - min_arr);
            swap(temp, a[--L[index]]);
            count++;
        }
    }

```

7. insertion-sort:

```cpp
void insertion_sort(vector<int>& arr) {
    int n = arr.size();
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}


/*
```

8. shaker-sort:

```
193
194  void shaker_sort(vector<int>& arr) {
195      int n = arr.size();
196      int left = 0;
197      int right = n - 1;
198      int temp;
199
200      while (left < right) {
201          for (int i = left; i < right; i++) {
202              if (arr[i] > arr[i + 1]) {
203                  temp = arr[i];
204                  arr[i] = arr[i + 1];
205                  arr[i + 1] = temp;
206              }
207          }
208          right--;
209
210          for (int i = right; i > left; i--) {
211              if (arr[i] < arr[i - 1]) {
212                  temp = arr[i];
213                  arr[i] = arr[i - 1];
214                  arr[i - 1] = temp;
215              }
216          }
217          left++;
218      }
219  }
220
```

9. heap-sort:

```
222
223  void Heapify(vector<int>& arr, int n, int i) {
224      int largest = i;
225      int left = 2 * i + 1;
226      int right = 2 * i + 2;
227
228      if (left < n && arr[left] > arr[largest])
229          largest = left;
230      if (right < n && arr[right] > arr[largest])
231          largest = right;
232
233      if (largest != i) {
234          swap(arr[i], arr[largest]);
235          Heapify(arr, n, largest);
236      }
237  }
238
239  void heap_sort(vector<int>& arr) {
240      int n = arr.size();
241
242      for (int i = n / 2 - 1; i >= 0; i--)
243          Heapify(arr, n, i);
244
245      for (int i = n - 1; i >= 1; i--) {
246          swap(arr[0], arr[i]);
247          Heapify(arr, i, 0);
248      }
249  }
250
```

10. quick-sort

```
251  // ---------------------------------------------------------------------
252
253  void quick_sort(vector<int>& arr, int low, int high) {
254      if (low < high) {
255          int pivot = arr[high];
256          int i = (low - 1);
257          for (int j = low; j < high; j++) {
258              if (arr[j] < pivot) {
259                  i++;
260                  swap(arr[i], arr[j]);
261              }
262          }
263
264          swap(arr[i + 1], arr[high]);
265          int pi = i + 1;
266
267          quick_sort(arr, low, pi - 1);
268          quick_sort(arr, pi + 1, high);
269      }
270  }
271
```

11. radix-sort:

```cpp

void radix_sort(vector<int>& arr) {
    int n = arr.size();
    if (n <= 1) return;

    int max_val = *max_element(arr.begin(), arr.end());

    for (int exp = 1; max_val / exp > 0; exp *= 10) {
        vector<int> output(n);
        int count[10] = {0};

        for (int i = 0; i < n; i++) {
            count[(arr[i] / exp) % 10]++;
        }

        for (int i = 1; i < 10; i++) {
            count[i] += count[i - 1];
        }

        for (int i = n - 1; i >= 0; i--) {
            int digit = (arr[i] / exp) % 10;
            output[count[digit] - 1] = arr[i];
            count[digit]--;
        }

        for (int i = 0; i < n; i++) {
            arr[i] = output[i];
        }
    }
}


```

12.main.cpp:

```cpp
// Hàm đọc dữ liệu từ file vào vector
void read_file(const string& name_file, vector<int>& data) {
    ifstream file(name_file);
    if (!file.is_open()) {
        cerr << "Khong the mo file: " << name_file << "!!!\n" << endl;
        exit(1);
    }
    int num;
    while (file >> num) {
        data.push_back(num);
    }
    file.close();
}

// Hàm ghi dữ liệu từ vector vào file
void write_file(const string& name_file, const vector<int>& data) {
    ofstream file(name_file);
    if (!file) {
        cerr << "Khong the mo file: " << name_file << "!!!\n" << endl;
        exit(1);
    }
    for (int num : data) {
        file << num << " ";
    }
    file.close();
}

int main(int argc, char* argv[]) {

    string algorithm, ip, op;
    vector<int> data;

    if (argc < 7) {
        cerr << "Cach su dung: " << argv[0] << " -a <ten thuat toan> -i <file input> -o <file output>" << endl;
        return 1;
    }

    // Xử lý tham số dòng lệnh
    for (int i = 1; i < argc; i++) {
        string arg = argv[i];
        if (arg == "-a" && i + 1 < argc) {
            algorithm = argv[++i];
        } else if (arg == "-i" && i + 1 < argc) {
            ip = argv[++i];
        } else if (arg == "-o" && i + 1 < argc) {
            op = argv[++i];
        } else {
            cerr << "Tham so khong hop le: " << arg << endl;
            return 1;
        }
    }

    // Đọc file đầu vào
    read_file(ip, data);

    // Kiểm tra nếu file rỗng
    if (data.empty()) {
        cerr << "File input rong!" << endl;
        return 1;
    }

```

```
71          // Chọn thuật toán sắp xếp
72          if (algorithm == "selection_sort") {
73              selection_sort(data);
74          } else if (algorithm == "bubble_sort") {
75              bubble_sort(data);
76          } else if (algorithm == "merge_sort") {
77              merge_sort(data, 0, data.size() - 1);
78          } else if (algorithm == "shell_sort") {
79              shell_sort(data);
80          } else if (algorithm == "counting_sort") {
81              counting_sort(data);
82          } else if (algorithm == "flash_sort") {
83              flash_sort(data);
84          } else if (algorithm == "insertion_sort") {
85              insertion_sort(data);
86          } else if (algorithm == "shaker_sort") {
87              shaker_sort(data);
88          } else if (algorithm == "heap_sort") {
89              heap_sort(data);
90          } else if (algorithm == "quick_sort") {
91              quick_sort(data, 0, data.size()- 1);
92          } else if (algorithm == "radix_sort") {
93              radix_sort(data);
94          } else {
95              cerr << "Thuat toan khong hop le!" << endl;
96              return 1;
97          }
98
99          // Ghi kết quả vào file
100         write_file(op, data);
101         cout << "\nDa sap xep xong, luu vao file " << op << "\n" <<endl;
102         return 0;
103     }
```