**1. Selection Sort**

- Thuật toán: duyệt mảng có n phần tử và chọn ra phần tử nhỏ nhất trong mảng và xếp phần tử đó lên đầu, cứ như vậy cho đến khi mảng được sắp xếp.

- Tốc độ:

- source code:

void selectionSort(int\* a, int n)

{

int minIdx;

for (int i = 0; i < n - 1; i++)

{

minIdx = i;

for (int j = i + 1; j < n; j++)

{

if (a[j] < a[minIdx])

{

minIdx = j;

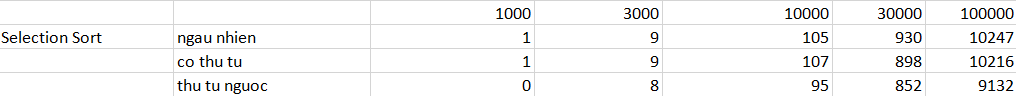
}

}

HoanVi(a[minIdx], a[i]);

}

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.  


**2. Bubble Sort**

- Thuật toán: so sánh 1 cặp phần tử ở cuối mảng, sau đó di chuyển phần tử nhỏ hơn lên trước, cứ như vậy cho đến hết vòng lặp ta được mảng đã sắp xếp giảm dần.

- Tốc độ:

- source code:

void bubbleSort(int\* a, int n)

{

for (int i = 1; i <n; i++)

{

for (int j = n - 1; j >= i; j--)

{

if (a[j] < a[j - 1])

{

HoanVi(a[j], a[j - 1]);

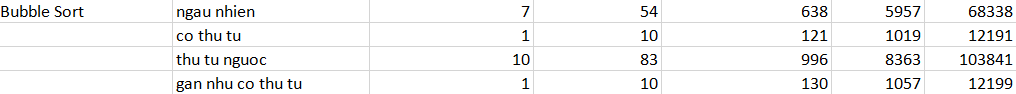
}

}

}

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



**3. Shaker Sort**

- Thuật toán: tiến bộ hơn Bubble Sort, Shaker Sort giúp đưa phần tử nhỏ nhất và đầu dãy đồng thời đưa phần tử lớn nhất về cuối dãy, Shaker Sort còn có biến để đánh dấu dãy đã được sắp xếp để không phải kiểm tra lại lần nữa.

- Tốc độ:

- Source code:

void shakerSort(int \*a, int n) //best n, worst n2

{

int Left = 0;

int Right = n - 1;

int k = 0;

while (Left < Right)

{

for (int i = Left; i < Right; i++)

{

if (a[i] > a[i + 1])

{

HoanVi(a[i], a[i + 1]);

k = i;

}

}

Right = k;

for (int i = Right; i > Left; i--)

{

if (a[i] < a[i - 1])

{

HoanVi(a[i], a[i - 1]);

k = i;

}

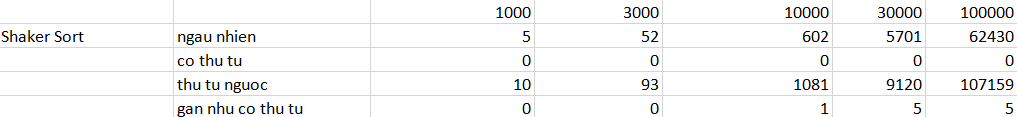
}

Left = k;

}

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



**4. Insertion Sort**

- Thuật toán: ý tưởng ở đây là chia mảng ra làm 2 phần, phần bên trái là phần được sắp xếp và phần bên phải là phần chưa được sắp xếp. Ta chọn lần lượt phần tử từ trái sang phải, so sánh phần tử được chọn với phần tử bên trái của nó, nếu phần tử được chọn nhỏ hơn, ta tiếp tục so sánh nó với phần tử tiếp theo bên trái cho đển khi phần tử được chọn lớn hơn hoặc bằng. Sau đó, dời các phần tử lớn hơn phần tủ được chọn qua bên phải 1 đơn vị và xếp phần tử được chọn vào đúng chỗ của nó ở mảng bên trái.

- Tốc độ:

- Source code:

void insertionSort(int\* a, int n)

{

int i, j, v;

for (int i = 1; i < n; i++)

{

v = a[i];

j = i - 1;

while (j >= 0 && a[j] > v) {

a[j + 1] = a[j];

j--;

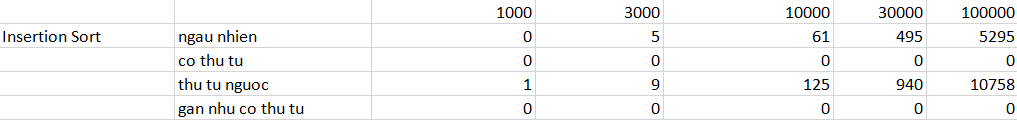
}

a[j + 1] = v;

}

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.

****

**5. Binary Insertion Sort**

- Thuật toán: cũng như Insertion Sort, các bước hoàn toàn giống nhau trừ phần tìm vị trí của phần tử được chọn ta dung hàm tiềm kiếm nhị phân để tối ưu hóa thời gian tiềm kiếm.

- Tốc độ:

- Source code:

void binaryInsertionSort(int\* a, int n)

{

int i, j, v;

for (int i = 1; i < n; i++)

{

v = a[i];

j = i - 1;

int search = binarySearch(a, j, a[i]);

while (j >= search)

{

a[j + 1] = a[j];

j--;

}

a[j + 1] = v;

}

}

int binarySearch(int\* a, int n, int key)

{

//gia bo cai array nay sap xep tang dan

int left = 0;

int right = n;

int mid = (left + right) / 2;

while (left < right)

{

if (a[mid] == key) return mid + 1;

if (a[mid] > key)

{

right = mid-1;

mid = (left + right) / 2;

}

else if (a[mid] < key)

{

left = mid+1;

mid = (left + right) / 2;

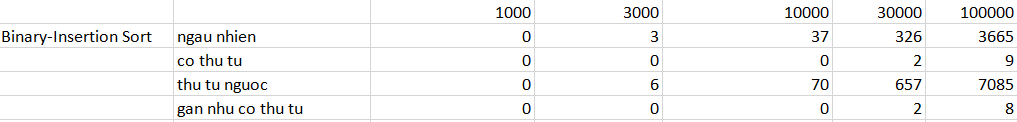
}

}

return key > a[left] ? left+1 : left;

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



**6. Quick Sort**

- Thuật toán: chọn ra 1 phần tử trong mảng (bên trái, bên phải hoặc ngẫu nhiên), ta dùng hàm Partition để tìm ra vị trí cuối cùng của phần tử được chọn sau khi sắp xếp và sắp nó vào vị trí đó, sau đó chia mảng ra làm 2 phần và tiếp tục làm như trên cho đến khi mảng còn 1 phần tử. Hàm Partition so sánh duyệt mảng từ 2 chiều bên trái và bên phải, nếu phần tử bên phải nhỏ hơn thì chuyển nó qua bên trái, nếu phần tử bên trái lớn hơn thì chuyển nó qua bên phải, cuối cùng vị trí giữa là vị trí cần tìm.

- Tốc độ:

- Source code:

int Partition(int\* a, int left, int right)

{

int p = a[(left+right)/2];

int i = left;

int j = right + 1;

do {

do i++; while (a[i] < p);

do j--; while (a[j] > p);

HoanVi(a[i], a[j]);

} while (i < j);

HoanVi(a[i], a[j]);

HoanVi(a[left], a[j]);

return j;

}

void quickSort(int\* a, int left, int right)

{

if (left < right)

{

int s = Partition(a, left, right); //s la vi tri cuoi cung cua a[left] trong mang sau khi sort

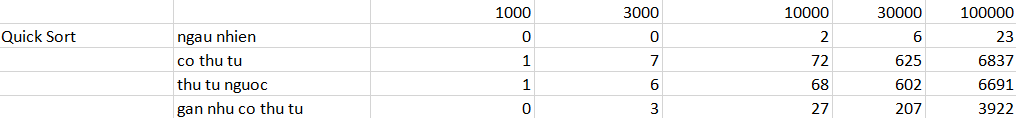
quickSort(a, left, s - 1);

quickSort(a, s+1, right);

}

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



**7. Heap Sort**

- Thuật toán: xây dựng 1 Max Heap, nghĩa là phần tử đầu tiên (vị trí i) sẽ là phần tử lớn nhất, và nó luôn lớn hơn phần tử ở vị trí bên trái (2i+1) và phần tử ở vị trí bên phải (2i+2). Đầu tiên ta xây dựng Max Heap bằng hàm Heapify với n phần tử, sau đó phần tử đầu tiên sẽ là phần tử lớn nhất, ta dịch chuyển phần tử đó xuống vị trí cuối cùng và thực hiện hàm Heapify với n-1 phần tử còn lại và tiếp tục tới khi hoàn thành.

- Tốc độ:

- Source code:

void heapify(int\* a, int n, int i)//logn

{

int max = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && a[left] > a[max])

{

max = left;

}

if (right < n && a[right] > a[max])

{

max = right;

}

if (max != i)

{

HoanVi(a[i], a[max]);

heapify(a, n, max);

}

}

void heapSort(int\* a, int n)

{

// Sap xep lai mang theo dang Heap

for (int i = n / 2 - 1; i >= 0; i--)

{

heapify(a, n, i);

}

// Sap xep theo thu tu tang dan

for (int i = n - 1; i >= 0; i--)

{

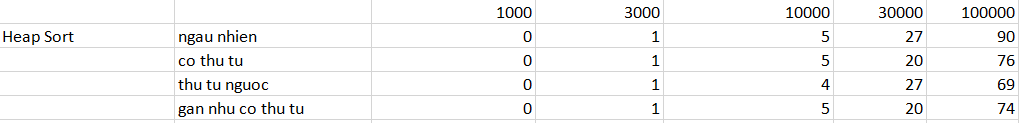
HoanVi(a[0], a[i]);

heapify(a, i, 0);

}

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



**8. Counting Sort**

- Thuật toán: sắp xếp cực nhanh không dựa vào phép so sánh mà dựa vào giá trị của phần tử trong mảng.

- Tốc độ:

- Source code:

void countingSort(int\* a, int n)

{

int b = findMax(a, n)+1;

int\* count = new int[b];

for (int i = 0; i < b; i++)

{

count[i] = 0;

}

//dem so lan xuat hien cua cac phan tu

for (int i = 0; i < n; i++)

{

count[a[i]]++;

}

for (int i = 1; i < b; i++)

{

count[i] += count[i - 1];

}

int\* temp = new int[n];

for (int i = 0; i < n; i++)

{

temp[count[a[i]] - 1] = a[i];

count[a[i]]--;

}

for (int i = 0; i < n; i++)

{

a[i]=temp[i];

}

delete []count, temp;

}

int findMax(int\* a, int n)

{

int max = 0;

for (int i = 0; i < n; i++)

{

if (a[i] > max)

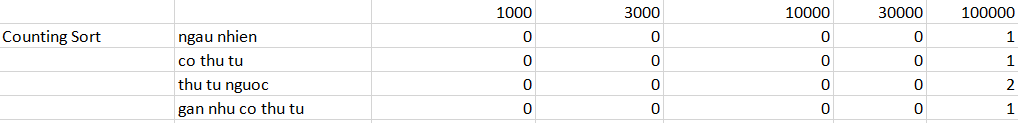
max = a[i];

}

return max;

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



**9. Merge Sort**

- Thuật toán: Là một thuật toán sắp xếp sử dựng phương pháp chia để trị. Nó chia mảng nhập vào thành hai phần rồi tiếp tục gọi đệ qui đến khi mảng chỉ còn một phần tử rồi bắt đầu ghép từng hai phần lại thành một mảng đã sắp xếp đến khi được một mảng hoàn chỉnh đã sắp xếp.

- Tốc độ:

- Source code:

void mergeSort(int\* &a, int n)

{

if (n == 1) return;

else

{

int sizeL = (n) / 2;

int sizeR = n - sizeL;

int\* L = new int[sizeL];

for (int i = 0; i < sizeL; i++)

{

L[i] = a[i];

}

int\* R = new int[sizeR];

for (int i = 0; i < sizeR; i++)

{

R[i] = a[i + sizeL];

}

mergeSort(L, sizeL);

mergeSort(R, sizeR);

merge(L, sizeL, R, sizeR, a);

}

}

void merge(int\* &L, int sizeL, int\* &R, int sizeR, int\* &a)

{

a = new int[sizeL + sizeR];

int i = 0, j = 0, k = 0;

while (i < sizeL && j < sizeR)

{

a[k++] = L[i] < R[j] ? L[i++] : R[j++];

}

while (i < sizeL)

{

a[k++] = L[i++];

}

while (j < sizeR)

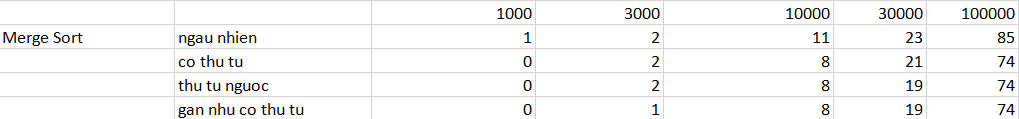
{

a[k++] = R[j++];

}

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



**10. Shell Sort**

- Thuật toán: Sử dụng Insertion Sort trên những khoảng gap = n/2 (với n là số phần tử trong mảng), rồi sau đó chia nhỏ những khoảng đó gap /= 2, gap /= 2… cho đến khi gap = 0. Giải thuật này tránh các trường hợp phải tráo đổi vị trí của hai phần tử xa nhau trong giải thuật sắp xếp chọn.

- Tốc độ:

- Source code:

void shellSort(int \*a, int n)

{

for (int gap = n / 2; gap > 0; gap /= 2)

{

for (int i = gap; i < n; i += 1)

{

int temp = a[i];

int j;

for (j = i; j >= gap && a[j - gap] > temp; j -= gap)

{

a[j] = a[j - gap];

}

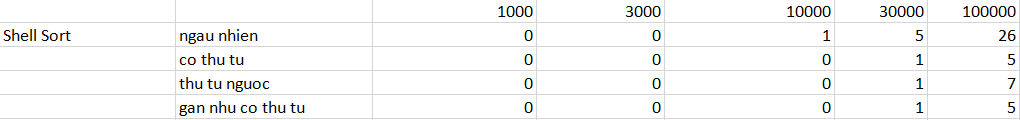
a[j] = temp;

}

}

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



**11. Radix Sort**

- Thuật toán: Thuật toán này được thực hiện dựa trên ý tưởng nếu một danh sách đã được sắp xếp hoàn chỉnh thì từng phần tử cũng sẽ được sắp xếp hoàn chỉnh dựa trên giá trị của các phần tử đó. Thuật toán sắp xếp phần tử theo hàng đơn vị, hàng chục, hàng trăm, hàng nghìn,.. và cứ như vậy cho đến khi mảng được sắp xếp hoàn thành

- Tốc độ:

- Source code:

void radixSort(int\* a, int n)

{

int k = findMax(a, n);

int exp = 1;

for (; k / exp > 0; exp \*= 10)

{

sort(a, n, exp);

}

}

void sort(int\* a, int n, int exp)

{

int count[10] = { 0 };

for (int i = 0; i < n; i++)

{

count[a[i] / exp % 10] ++;

}

for (int i = 1; i < 10; i++)

{

count[i] += count[i - 1];

}

int\* b = new int[n];

for (int i = n - 1; i >= 0; i--)

{

b[count[a[i] / exp % 10] - 1] = a[i];

count[a[i] / exp % 10] --;

}

for (int i = 0; i <n; i++)

{

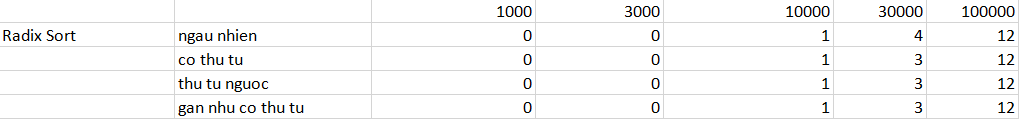
a[i] = b[i];

}

delete[]b;

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



**12. Flash Sort**

- Thuật toán: Flash Sort gồm 3 bước. Bước 1: chia phân lớp, duyệt mảng 1 lượt để xác định các phần tử thuộc phân lớp nào bằng công thức có sẵn. Bước 2: di chuyển các phần tử về đúng phân lớp của mình. Bước 3: sắp xếp, nếu phân lớp đó còn quá nhiều phần tử ta dùng Flash Sort để làm tiếp, nếu phân lớp có ít phần tử ta sử dụng Insertion Sort cho nhanh.

- Tốc độ:

- Source code:

void flashSort(int \*a, int n)

{

if (n == 0) return;

int m = (int)((0.2 \* n) + 2); //so phan lop

//CLASS FORMATION O(n)

int min, max, maxIndex = 0;

min = max = a[0];

for (int i = 1; i < n; i++)

{

if (a[i] < min) min = a[i];

else if (a[i] > max)

{

max = a[i];

maxIndex = i;

}

}

if (max == min)

{

return; //all the elements are the same

}

int\* L = new int[m + 1]; //khong su dung L[0], L la mang chua so phan tu cua tung phan lop

for (int i = 1; i < m + 1; i++)

{

L[i] = 0;

}

int K; //K thuoc 1...m

for (int i = 0; i < n; i++)

{

K = 1 + (int)((m - 1) \* (a[i] - min) / (max - min));

L[K] += 1;

}

for (K = 2; K <= m; K++)

{

L[K] = L[K] + L[K - 1];

}

//-------PERMUTATION-------

HoanVi(a[maxIndex], a[0]);

int j = 0, numMoves = 0;

K = m;

while (numMoves < n)

{

while (j >= L[K])

{

j++;

K = 1 + (int)((m - 1) \* (a[j] - min) / (max - min));

}

int evicted = a[j];

while (j < L[K])

{

K = 1 + (int)((m - 1) \* (evicted - min) / (max - min));

int location = L[K] - 1;

HoanVi(a[location], evicted);

L[K] --;

numMoves++;

}

}

//-------RECURSION or STRAIGHT INSERTION-------

int threshold = (int)(1.25 \* ((n / m) + 1));

const int minE = 30;

for (K = m - 1; K >= 1; K--)

{

int classSize = L[K + 1] - L[K];

if (classSize > threshold && classSize > minE)

{

flashSort(&a[L[K]], classSize);

}

else

{

if (classSize > 1) insertionSort(&a[L[K]], classSize);

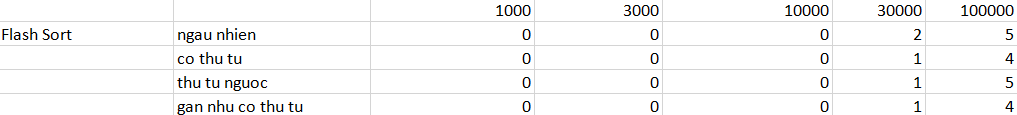
}

}

delete[] L;

}

Ví dụ về tốc độ của thuật toán với n=1000,3000,10000,30000,100000, thời gian tính bằng milliseconds.



Bảng so sánh tốc độ của 12 thuật toán Sort trên các 4 kiểu sắp xếp khác nhau:

