Отчет по лабораторной работе номер 5  
«Сортировки»

Выполнил студент группы J3113  
Бочаров Евгений

465274

1. Сортировка пузырьком

void bubbleSort(std::vector<int>& arr) { // Сортировка пузырьком, сложность O(N^2), память O(1)

int n = arr.size();

bool swapped; // O(1) - используемая память для флага

for (int i = 0; i < n - 1; i++) { // O(N)

swapped = false; // O(1)

for (int j = 0; j < n - i - 1; j++) { //O(N)

if (arr[j] > arr[j + 1]) {

std::swap(arr[j], arr[j + 1]); // O(1)

swapped = true; // O(1)

}

}

if (!swapped) break;

}

}

1. Merge sort

void merge(std::vector<int>& arr, int left, int mid, int right) { // Слияние двух подмассивов, сложность O(N), память O(N)

int n1 = mid - left + 1;

int n2 = right - mid;

std::vector<int> L(n1), R(n2); // O(n1 + n2) - использованная память

for (int i = 0; i < n1; i++)

L[i] = arr[left + i]; // O(n1)

for (int j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j]; // O(n2)

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

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i]; // O(1)

i++;

}

else {

arr[k] = R[j]; // O(1)

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i]; // O(1)

i++;

k++;

}

while (j < n2) {

arr[k] = R[j]; // O(1)

j++;

k++;

}

}

void mergeSort(std::vector<int>& arr, int left, int right) { // Основная функция Merge Sort, сложность O(N\*logN), паямть O(N)

if (left < right) {

int mid = left + (right - left) / 2; // O(1)

mergeSort(arr, left, mid); // Рекурсивный вызов

mergeSort(arr, mid + 1, right); // Рекурсивный вызов

merge(arr, left, mid, right); // Слияние подмассивов

}

}

1. Timsort

void merge(std::vector<int>& arr, int left, int mid, int right) { // Слияние двух подмассивов, сложность O(N), память O(N)

int n1 = mid - left + 1;

int n2 = right - mid;

std::vector<int> L(n1), R(n2); // O(n1 + n2) - использованная память

for (int i = 0; i < n1; i++)

L[i] = arr[left + i]; // O(n1)

for (int j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j]; // O(n2)

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

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i]; // O(1)

i++;

}

else {

arr[k] = R[j]; // O(1)

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i]; // O(1)

i++;

k++;

}

while (j < n2) {

arr[k] = R[j]; // O(1)

j++;

k++;

}

}

void insertionSort(std::vector<int>& arr, int left, int right) { // Функция для сортировки подмассива, сложность O(N), память O(1)

for (int i = left + 1; i <= right; i++) {

int key = arr[i]; // O(1) - используемая память

int j = i - 1;

while (j >= left && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

void timSort(std::vector<int>& arr) { // Основная функция Timsort, сложность O(N\*logN), память O(N)

int RUN = 32;

int n = arr.size();

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

insertionSort(arr, i, std::min(i + RUN - 1, n - 1)); // O(n) для вставки

}

for (int size = RUN; size < n; size \*= 2) {

for (int left = 0; left < n; left += 2 \* size) {

int mid = std::min(left + size - 1, n - 1);

int right = std::min((left + 2 \* size - 1), (n - 1));

if (mid < right)

merge(arr, left, mid, right); // O(n) для слияния

}

}

}

Линейный график зависимости времени выполнения сортировок от размера массива

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

Вывод: ассимптотика с практикой соотносится хорошо, выбросы есть из-за разницы в даннх, разных случаях.

Bubble sort подходит разве что для образовательных целей.

Merge sort хорошо подходит для связных списков и случаев, когда важна стабильность сортировки(равные элементы сохраняют свой порядок)

Timsort эффективен в почти всегда, поэтому и используется в качестве встроенной сортировки в некторых ЯП.