# Сравнение insert()

## Vector.insert()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 2 | 36 | 371 |
| 2 | 2 | 49 | 407 |
| 3 | 3 | 39 | 595 |
| 4 | 3 | 60 | 375 |
| 5 | 3 | 32 | 387 |
| 6 | 2 | 27 | 354 |
| 7 | 3 | 47 | 348 |
| 8 | 3 | 56 | 380 |
| 9 | 2 | 31 | 489 |
| 10 | 2 | 31 | 399 |
| average value | 2,5 | 40,8 | 410,5 |

## List.insert()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 8 | 119 | 1143 |
| 2 | 7 | 116 | 887 |
| 3 | 9 | 87 | 744 |
| 4 | 8 | 86 | 1187 |
| 5 | 8 | 89 | 1232 |
| 6 | 8 | 96 | 1166 |
| 7 | 7 | 107 | 1204 |
| 8 | 7 | 92 | 1416 |
| 9 | 15 | 88 | 1173 |
| 10 | 7 | 94 | 1337 |
| average value | 8,4 | 97,4 | 1148,9 |

## Вывод

Метод insert предназначен для заполнения контейнера значениями. В тестировании значения были рандомными, по графику можно сделать вывод,что на любых объемах памяти std::vector справляется медленнее, чем std::list

## График

## Листинг

void insertvctr(std::vector<int>& a, int N)

{

auto begin = std::chrono::steady\_clock::now();

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

{

a[i] = rand() % 100;

//std::cout << a[i] << ",";

}

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}

void insert\_lst(std::list<int>& a, int N)

{

auto begin = std::chrono::steady\_clock::now();

list<int>::iterator it = a.begin();

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

{

a.insert(it,rand() % 100);

}

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}

# Сравнение sort()

## Vector.sort()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 2 | 27 | 267 |
| 2 | 2 | 27 | 259 |
| 3 | 3 | 26 | 251 |
| 4 | 2 | 41 | 280 |
| 5 | 2 | 26 | 437 |
| 6 | 2 | 26 | 246 |
| 7 | 2 | 25 | 246 |
| 8 | 2 | 25 | 246 |
| 9 | 2 | 25 | 245 |
| 10 | 2 | 25 | 245 |
| average value | 2,1 | 27,3 | 272,2 |

## List.sort()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 18 | 435 | 5666 |
| 2 | 16 | 815 | 5696 |
| 3 | 19 | 344 | 6144 |
| 4 | 17 | 806 | 5742 |
| 5 | 16 | 585 | 5645 |
| 6 | 16 | 778 | 5562 |
| 7 | 17 | 723 | 5614 |
| 8 | 17 | 680 | 5645 |
| 9 | 30 | 763 | 6100 |
| 10 | 17 | 696 | 5650 |
| average value | 18,3 | 662,5 | 5746,4 |

## Вывод

Исходя из полученных данных, можно сделать вывод, что при сортировке std::vector справляется намного лучше,чем std::list. Данные представлены на графике

## График

## Листинг

void sort\_vctr(std::vector<int>& a, int N)

{

auto begin = std::chrono::steady\_clock::now();

std::sort(a.begin(), a.end());

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}

void sort\_lst(std::list<int>& a, int N)

{

auto begin = std::chrono::steady\_clock::now();

a.sort();

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}

# Сравнение pop.back()

## Vector.pop\_back();

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 0 | 1 | 36 |
| 2 | 0 | 1 | 78 |
| 3 | 0 | 1 | 18 |
| 4 | 1 | 7 | 24 |
| 5 | 0 | 1 | 23 |
| 6 | 0 | 1 | 34 |
| 7 | 0 | 13 | 51 |
| 8 | 0 | 1 | 22 |
| 9 | 0 | 1 | 22 |
| 10 | 0 | 1 | 42 |
| average value | 0,1 | 2,8 | 35 |

## List.pop\_back();

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 6 | 68 | 735 |
| 2 | 7 | 84 | 819 |
| 3 | 5 | 62 | 1046 |
| 4 | 6 | 85 | 791 |
| 5 | 5 | 90 | 984 |
| 6 | 5 | 142 | 693 |
| 7 | 6 | 78 | 690 |
| 8 | 5 | 78 | 729 |
| 9 | 6 | 94 | 759 |
| 10 | 7 | 96 | 787 |
| average value | 5,8 | 87,7 | 803,3 |

## Вывод

Метод pop\_back() применяется для удаления значения в конце вектора/списка. По графику видно, что std::vector затрачивает меньше времени на данную операцию.

## График

## Листинг

void vctr\_pop\_back(std::vector<int>& a, int c)

{

auto begin = std::chrono::steady\_clock::now();

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

{

a.pop\_back();

}

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}

void pop\_backlst(std::list<int>& a, int c)

{

auto begin = std::chrono::steady\_clock::now();

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

{

a.pop\_back();

}

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}

# Сравнение push \_back() и push \_front() в list

## List.push\_front()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 12 | 210 | 3220 |
| 2 | 12 | 322 | 2860 |
| 3 | 13 | 226 | 2228 |
| 4 | 15 | 195 | 2408 |
| 5 | 11 | 211 | 2342 |
| 6 | 12 | 181 | 2406 |
| 7 | 13 | 212 | 2507 |
| 8 | 12 | 241 | 2470 |
| 9 | 10 | 170 | 2227 |
| 10 | 11 | 220 | 2253 |
| average value | 12,1 | 218,8 | 2492,1 |

## list.push\_back()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 15 | 229 | 1661 |
| 2 | 13 | 173 | 2803 |
| 3 | 12 | 185 | 2365 |
| 4 | 14 | 170 | 2399 |
| 5 | 12 | 202 | 2434 |
| 6 | 12 | 179 | 2354 |
| 7 | 12 | 212 | 2336 |
| 8 | 18 | 205 | 2170 |
| 9 | 11 | 241 | 2546 |
| 10 | 11 | 237 | 2658 |
| average value | 13 | 203,3 | 2372,6 |

## Вывод

В контейнере std::list добавлять элементы можно и из начала, и с конца, поскольку список реализован как двусвязный. По графику видно, что добавление с конца чуть более выгодно, однако разница незначительна.

## График

## Листинг

void push\_backlist(std::list<int>& a, int N)

{

auto begin = std::chrono::steady\_clock::now();

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

{

a.push\_back(rand() % 100);

// a.pop\_front();

}

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}

void push\_frontlist(std::list<int>& a, int N)

{

auto begin = std::chrono::steady\_clock::now();

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

{

a.push\_front(rand() % 100);

// a.pop\_back();

}

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}

# Сравнение push\_back()

## Vector.push\_back()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 2 | 46 | 459 |
| 2 | 3 | 29 | 278 |
| 3 | 2 | 31 | 416 |
| 4 | 3 | 28 | 428 |
| 5 | 3 | 27 | 471 |
| 6 | 2 | 29 | 647 |
| 7 | 3 | 32 | 475 |
| 8 | 3 | 31 | 403 |
| 9 | 2 | 28 | 391 |
| 10 | 3 | 31 | 433 |
| average value |  |  |  |

## list.push\_back()

|  |  |  |  |
| --- | --- | --- | --- |
|  | 90000 int, ms | 900000 int, ms | 9000000 int, ms |
| 1 | 15 | 229 | 1661 |
| 2 | 13 | 173 | 2803 |
| 3 | 12 | 185 | 2365 |
| 4 | 14 | 170 | 2399 |
| 5 | 12 | 202 | 2434 |
| 6 | 12 | 179 | 2354 |
| 7 | 12 | 212 | 2336 |
| 8 | 18 | 205 | 2170 |
| 9 | 11 | 241 | 2546 |
| 10 | 11 | 237 | 2658 |
| average value | 13 | 203,3 | 2372,6 |

## Вывод

В контейнере std::vector добавление в конец происходит значительно быстрее, нежели в std::list

## График

## Листинг

void vctr\_push\_back(std::vector<int>& a, int N)

{

auto begin = std::chrono::steady\_clock::now();

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

{

int num = rand() % 100;

a.push\_back(num);

}

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}

void push\_backlist(std::list<int>& a, int N)

{

auto begin = std::chrono::steady\_clock::now();

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

{

a.push\_back(rand() % 100);

// a.pop\_front();

}

auto end = std::chrono::steady\_clock::now();

auto elapsed\_ms = std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin);

std::cout << "The time: " << elapsed\_ms.count() << " ms\n";

}