


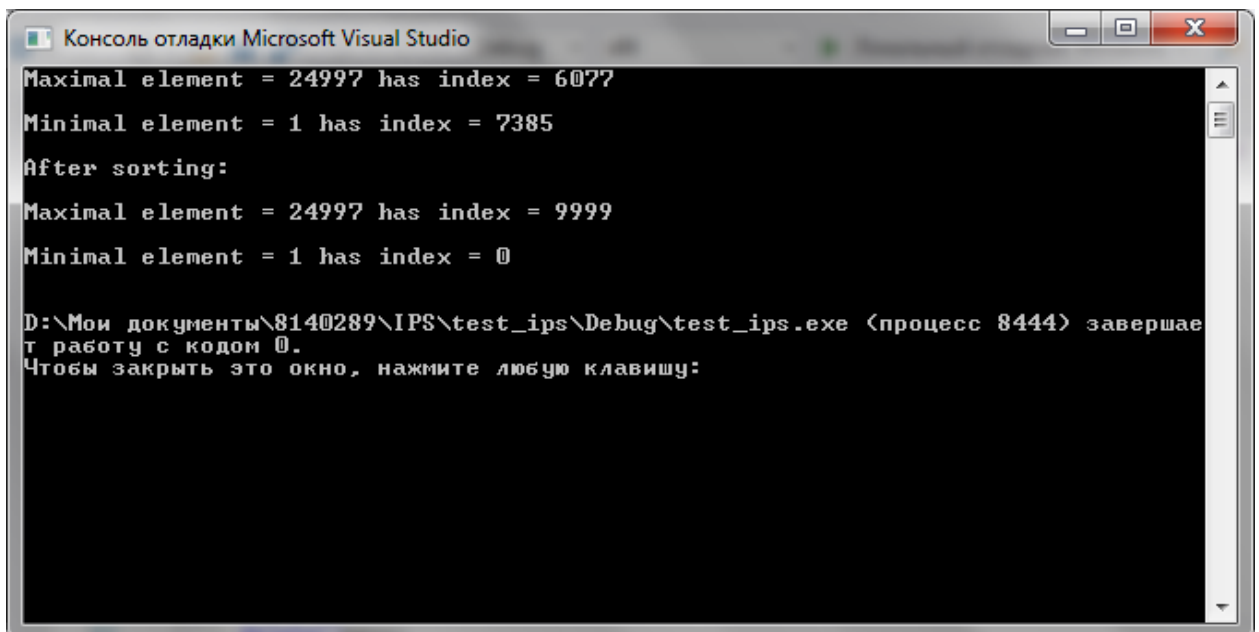
Задание к занятию 2

1. Разберите пример программы нахождения максимального элемента массива и его индекса [task for lecture2.cpp](#) . Запустите программу и убедитесь в корректности ее работы.

Программа работает корректно, после запуска выводится информация о максимальном элементе массива и его индексе до и после сортировки массива. Элемент найден верно (один и тот же), индекс элемента после сортировки равен размеру массива минус 1.

2. По аналогии с функцией **ReducerMaxTest(...)**, реализуйте функцию **ReducerMinTest(...)** для нахождения минимального элемента массива и его индекса. Вызовите функцию **ReducerMinTest(...)** до сортировки исходного массива **mass** и после сортировки. Убедитесь в правильности работы функции **ParallelSort(...)**: индекс минимального элемента после сортировки должен быть равен **0**, индекс максимального элемента (**mass_size - 1**).

Функция работает корректно, индекс минимального элемента после сортировки равен 0.

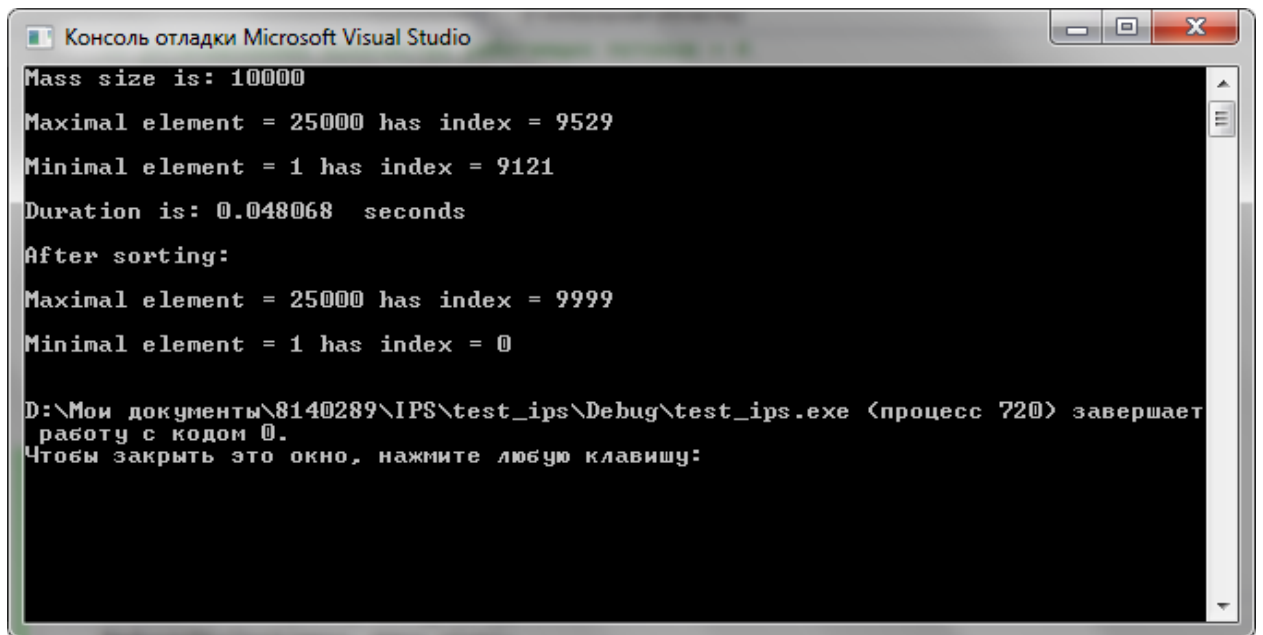


```
Консоль отладки Microsoft Visual Studio
Maximal element = 24997 has index = 6077
Minimal element = 1 has index = 7385
After sorting:
Maximal element = 24997 has index = 9999
Minimal element = 1 has index = 0

D:\Мои документы\8140289\IPS\test_ips\Debug\test_ips.exe (процесс 8444) завершает
работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
```

3. Добавьте в функцию **ParallelSort(...)** строки кода для измерения времени, необходимого для сортировки исходного массива. Увеличьте количество элементов **mass_size** исходного массива **mass** в **10, 50, 100** раз по сравнению с первоначальным. Выводите в консоль время, затраченное на сортировку массива, для каждого из значений **mass_size**. **Рекомендуется** засекаать время с помощью библиотеки **chrono**.

Для исходного размера массива:

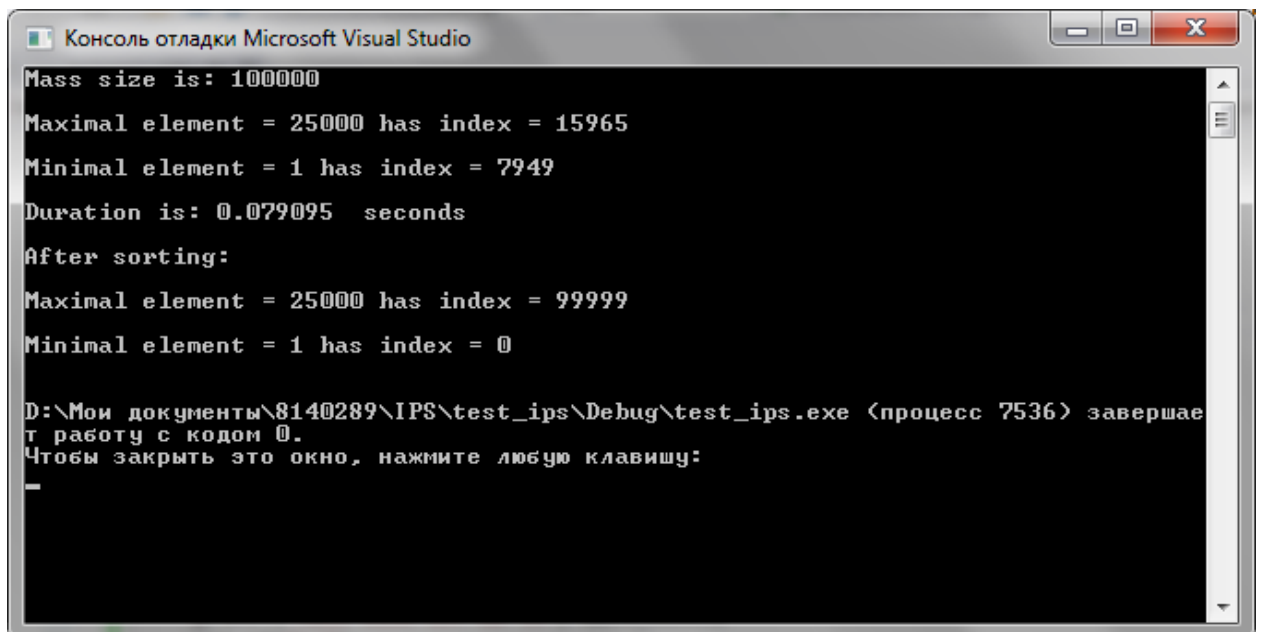


```
Консоль отладки Microsoft Visual Studio

Mass size is: 10000
Maximal element = 25000 has index = 9529
Minimal element = 1 has index = 9121
Duration is: 0.048068 seconds
After sorting:
Maximal element = 25000 has index = 9999
Minimal element = 1 has index = 0

D:\Мои документы\8140289\IPS\test_ips\Debug\test_ips.exe (процесс 720) завершает
работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
```

*Для размера массива *10:*

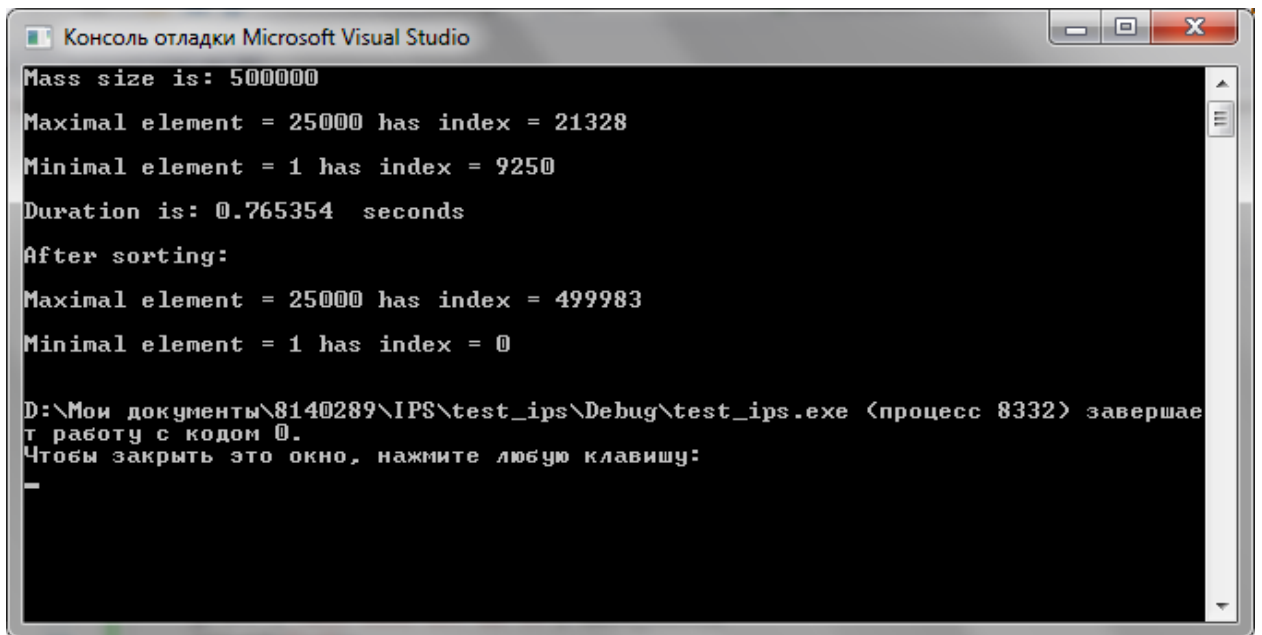


```
Консоль отладки Microsoft Visual Studio

Mass size is: 100000
Maximal element = 25000 has index = 15965
Minimal element = 1 has index = 7949
Duration is: 0.079095 seconds
After sorting:
Maximal element = 25000 has index = 99999
Minimal element = 1 has index = 0

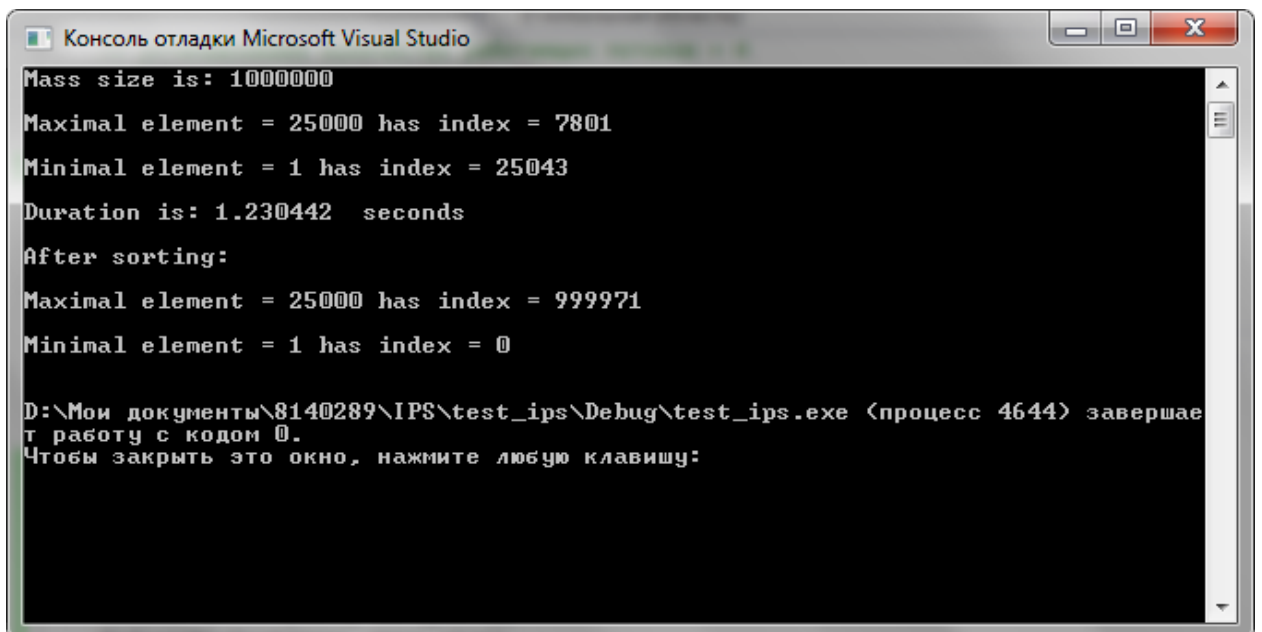
D:\Мои документы\8140289\IPS\test_ips\Debug\test_ips.exe (процесс 7536) завершае
т работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
-
```

*Для размера массива *50:*



```
Консоль отладки Microsoft Visual Studio
Mass size is: 500000
Maximal element = 25000 has index = 21328
Minimal element = 1 has index = 9250
Duration is: 0.765354 seconds
After sorting:
Maximal element = 25000 has index = 499983
Minimal element = 1 has index = 0
D:\Мои документы\8140289\IPS\test_ips\Debug\test_ips.exe (процесс 8332) завершае
т работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
-
```

Для размера массива *100:



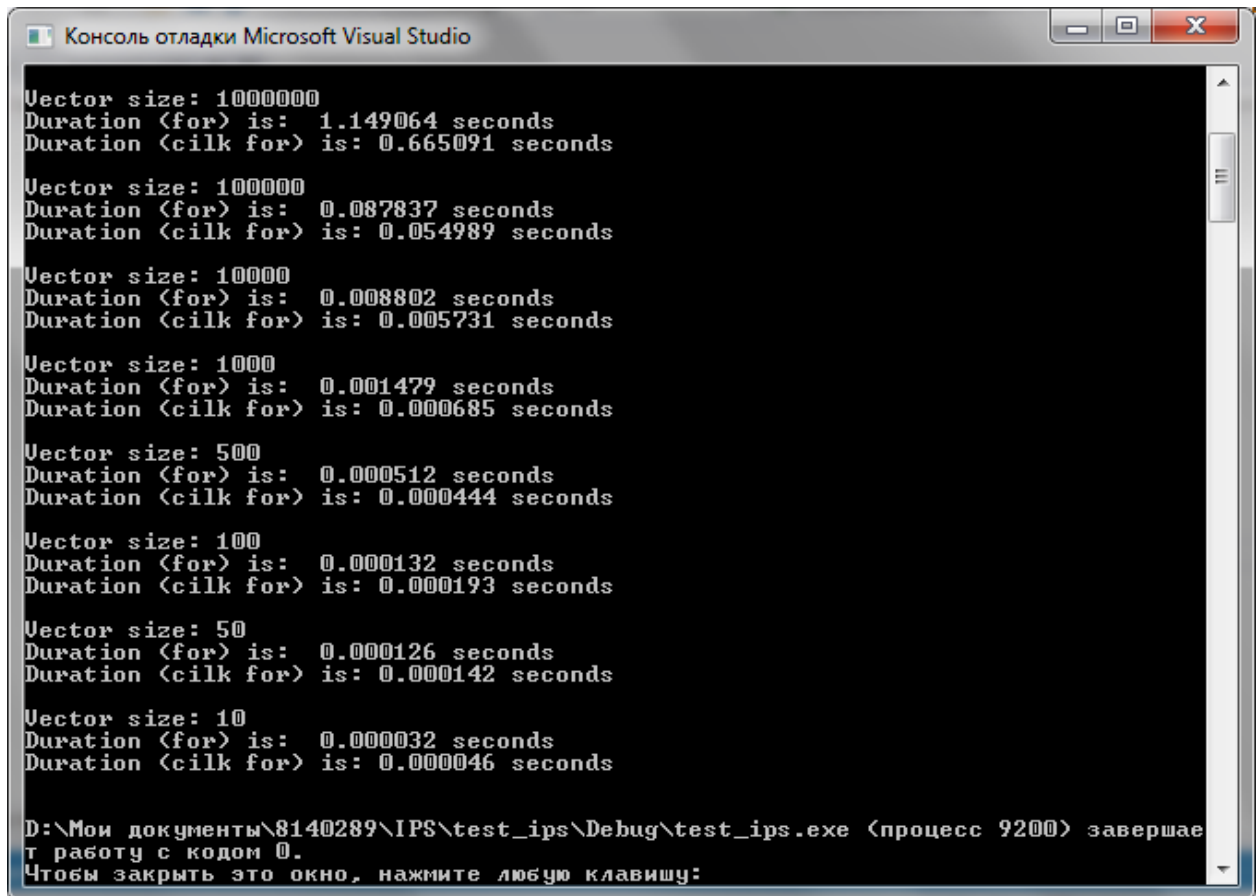
```
Консоль отладки Microsoft Visual Studio
Mass size is: 1000000
Maximal element = 25000 has index = 7801
Minimal element = 1 has index = 25043
Duration is: 1.230442 seconds
After sorting:
Maximal element = 25000 has index = 999971
Minimal element = 1 has index = 0
D:\Мои документы\8140289\IPS\test_ips\Debug\test_ips.exe (процесс 4644) завершае
т работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
```

Видно, что в последних двух случаях индекс максимального элемента в сортированном массиве не равен (**mass_size - 1**). Это может быть из-за того, что начиная с этого индекса, все элементы - максимальные (то есть равны).

4. Реализуйте функцию **CompareForAndCilk_For(size_t sz)**. Эта функция должна выводить на консоль время работы стандартного цикла **for**, в котором заполняется случайными значениями **std::vector** (использовать функцию **push_back(rand() % 20000 + 1)**), и время работы параллельного цикла **cilk_for** от **Intel Cilk Plus**, в котором заполняется случайными значениями **reducer вектор**.

Параметр функции **sz** - количество элементов в каждом из векторов.

Вызывайте функцию **CompareForAndCilk_For()** для входного параметра **sz** равного: **1000000, 100000, 10000, 1000, 500, 100, 50, 10**. Проанализируйте результаты измерения времени, необходимого на заполнение **std::vector**'а и **reducer** вектора.



```
Vector size: 1000000
Duration <for> is: 1.149064 seconds
Duration <cilk for> is: 0.665091 seconds

Vector size: 100000
Duration <for> is: 0.087837 seconds
Duration <cilk for> is: 0.054989 seconds

Vector size: 10000
Duration <for> is: 0.008802 seconds
Duration <cilk for> is: 0.005731 seconds

Vector size: 1000
Duration <for> is: 0.001479 seconds
Duration <cilk for> is: 0.000685 seconds

Vector size: 500
Duration <for> is: 0.000512 seconds
Duration <cilk for> is: 0.000444 seconds

Vector size: 100
Duration <for> is: 0.000132 seconds
Duration <cilk for> is: 0.000193 seconds

Vector size: 50
Duration <for> is: 0.000126 seconds
Duration <cilk for> is: 0.000142 seconds

Vector size: 10
Duration <for> is: 0.000032 seconds
Duration <cilk for> is: 0.000046 seconds

D:\Мои документы\8140289\IPS\test_ips\Debug\test_ips.exe (процесс 9200) завершае
т работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
```

5. Ответьте на вопросы:

Почему при небольших значениях **sz** цикл **cilk_for** уступает циклу **for** в быстройдействии?

*Циклы **cilk_for** реализованы с использованием алгоритма "разделяй и властвуй", который рекурсивно разделяет диапазон пополам, пока количество оставшихся итераций меньше, чем "размер зерна". При маленьком значении размера вектора затрачивается больше времени на разбиение, чем экономится на параллельных вычислениях.*

В каких случаях целесообразно использовать цикл **cilk_for** ?

*Использование цикла **cilk_for** целесообразно при большом диапазоне счетчика.*

В чем принципиальное отличие параллелизации с использованием **cilk_for** от параллелизации с использованием **cilk_spawn** в паре с **cilk_sync**?

*Ключевое слово **spawn** - создаёт новую задачу, вызывая Cilk-процедуру, которая может выполняться параллельно, а **sync** - ожидает завершения ранее созданных задач и продолжает работу. Cilk-процедура не должна использовать*

значения, возвращаемые дочерними процедурами, пока не будет выполнен оператор `sync`. Если на момент вызова этого оператора хотя бы одна дочерняя процедура продолжает выполняться, родительская процедура приостанавливает своё выполнение и не продолжает его до тех пор, пока все задачи не завершатся. После этого, процедура продолжает своё выполнение.

Циклы `cilk_for` реализованы с использованием алгоритма, в котором диапазон рекурсивно разделяется пополам, процедуры выполняются параллельно и не ждут завершения других процедур.