
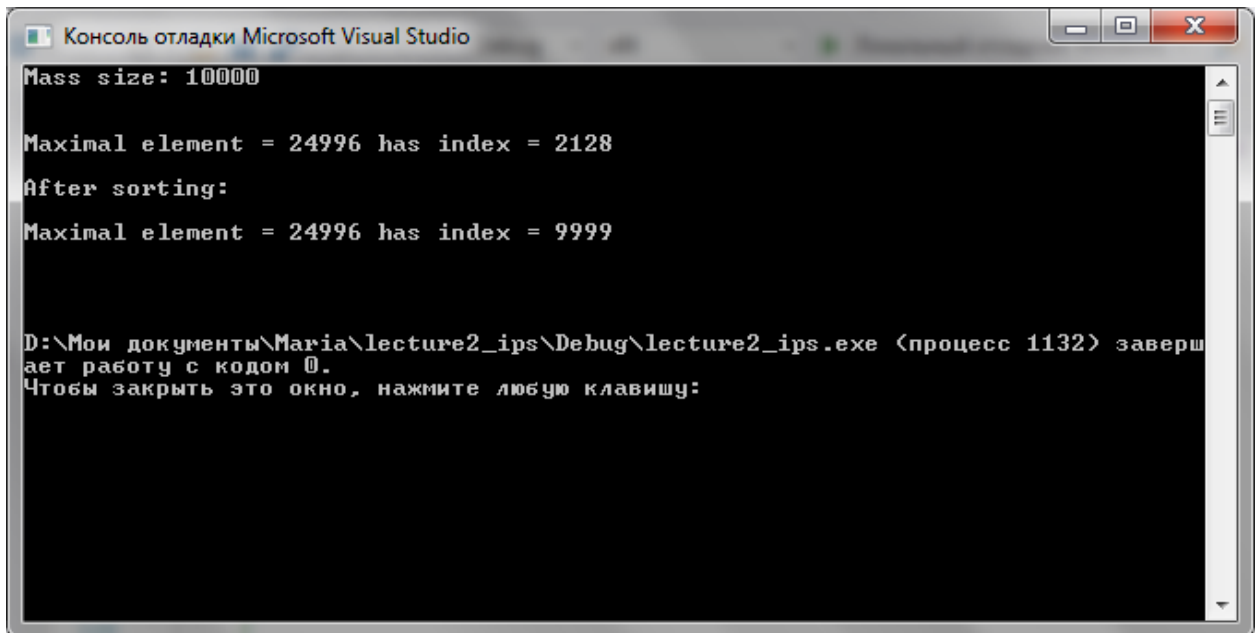


Выполнила: Шахманова Мария, ПМ-21м

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

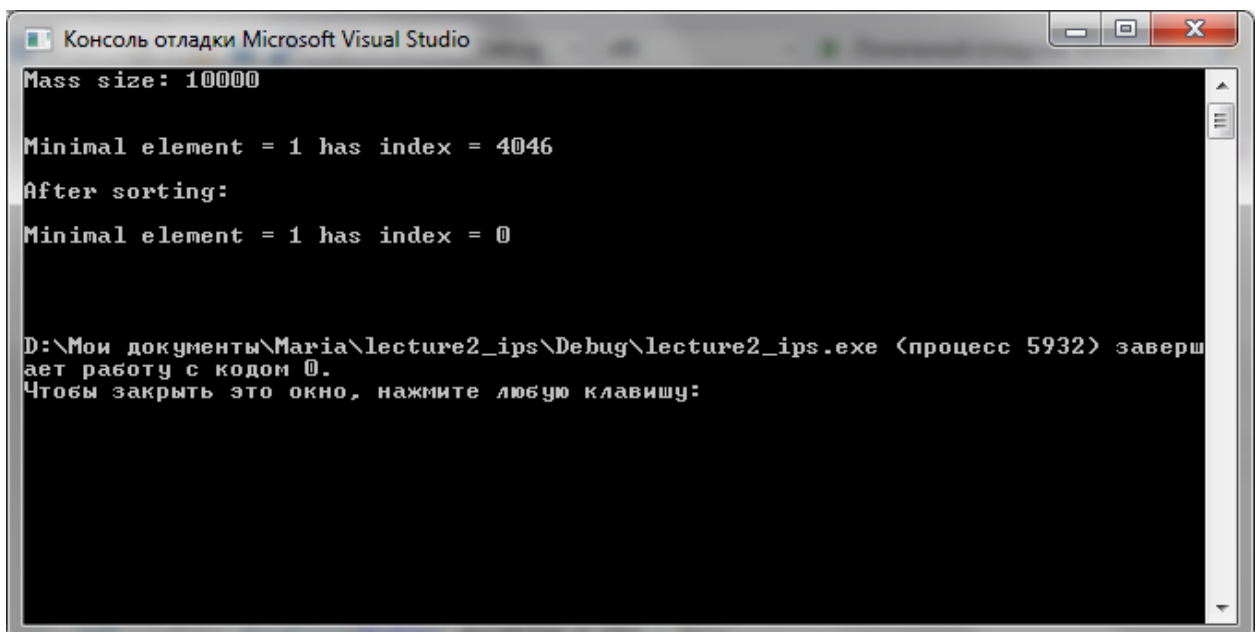


```
Консоль отладки Microsoft Visual Studio
Mass size: 10000
Maximal element = 24996 has index = 2128
After sorting:
Maximal element = 24996 has index = 9999

D:\Мои документы\Maria\lecture2_ips\Debug\lecture2_ips.exe (процесс 1132) заверш
ает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
```

Программа работает корректно.

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**).

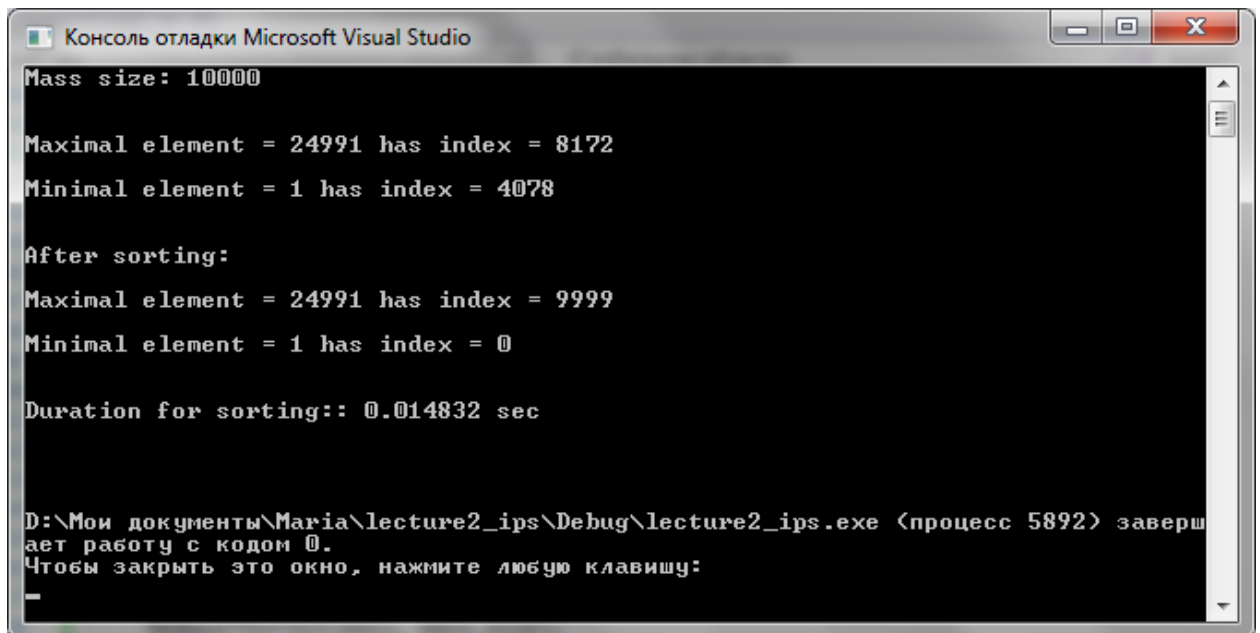


```
Консоль отладки Microsoft Visual Studio
Mass size: 10000
Minimal element = 1 has index = 4046
After sorting:
Minimal element = 1 has index = 0

D:\Мои документы\Maria\lecture2_ips\Debug\lecture2_ips.exe (процесс 5932) заверш
ает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
```

Программа работает корректно.

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***.



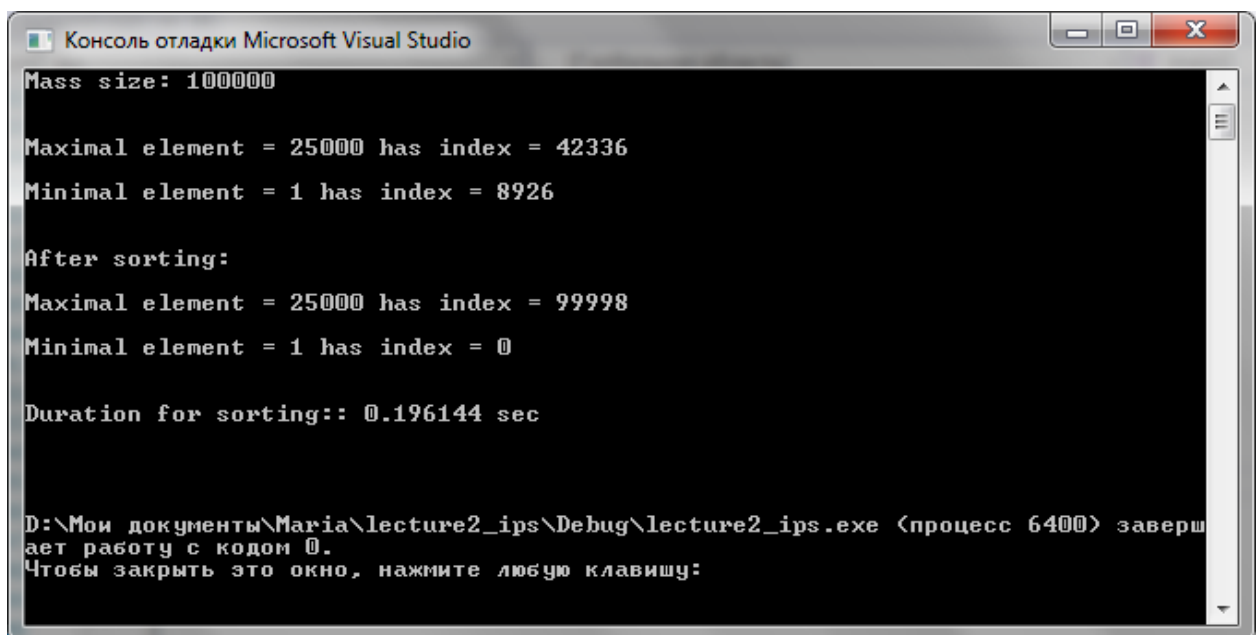
```
Консоль отладки Microsoft Visual Studio
Mass size: 10000

Maximal element = 24991 has index = 8172
Minimal element = 1 has index = 4078

After sorting:
Maximal element = 24991 has index = 9999
Minimal element = 1 has index = 0

Duration for sorting:: 0.014832 sec

D:\Мои документы\Maria\lecture2_ips\Debug\lecture2_ips.exe (процесс 5892) заверш
ает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
-
```



```
Консоль отладки Microsoft Visual Studio
Mass size: 100000

Maximal element = 25000 has index = 42336
Minimal element = 1 has index = 8926

After sorting:
Maximal element = 25000 has index = 99998
Minimal element = 1 has index = 0

Duration for sorting:: 0.196144 sec

D:\Мои документы\Maria\lecture2_ips\Debug\lecture2_ips.exe (процесс 6400) заверш
ает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
```

```
Консоль отладки Microsoft Visual Studio

Mass size: 500000

Maximal element = 25000 has index = 40650
Minimal element = 1 has index = 1921

After sorting:
Maximal element = 25000 has index = 499986
Minimal element = 1 has index = 0

Duration for sorting:: 0.824427 sec

D:\Мои документы\Maria\lecture2_ips\Debug\lecture2_ips.exe (процесс 8096) заверш
ает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
_
```

```
Консоль отладки Microsoft Visual Studio

Mass size: 10000000

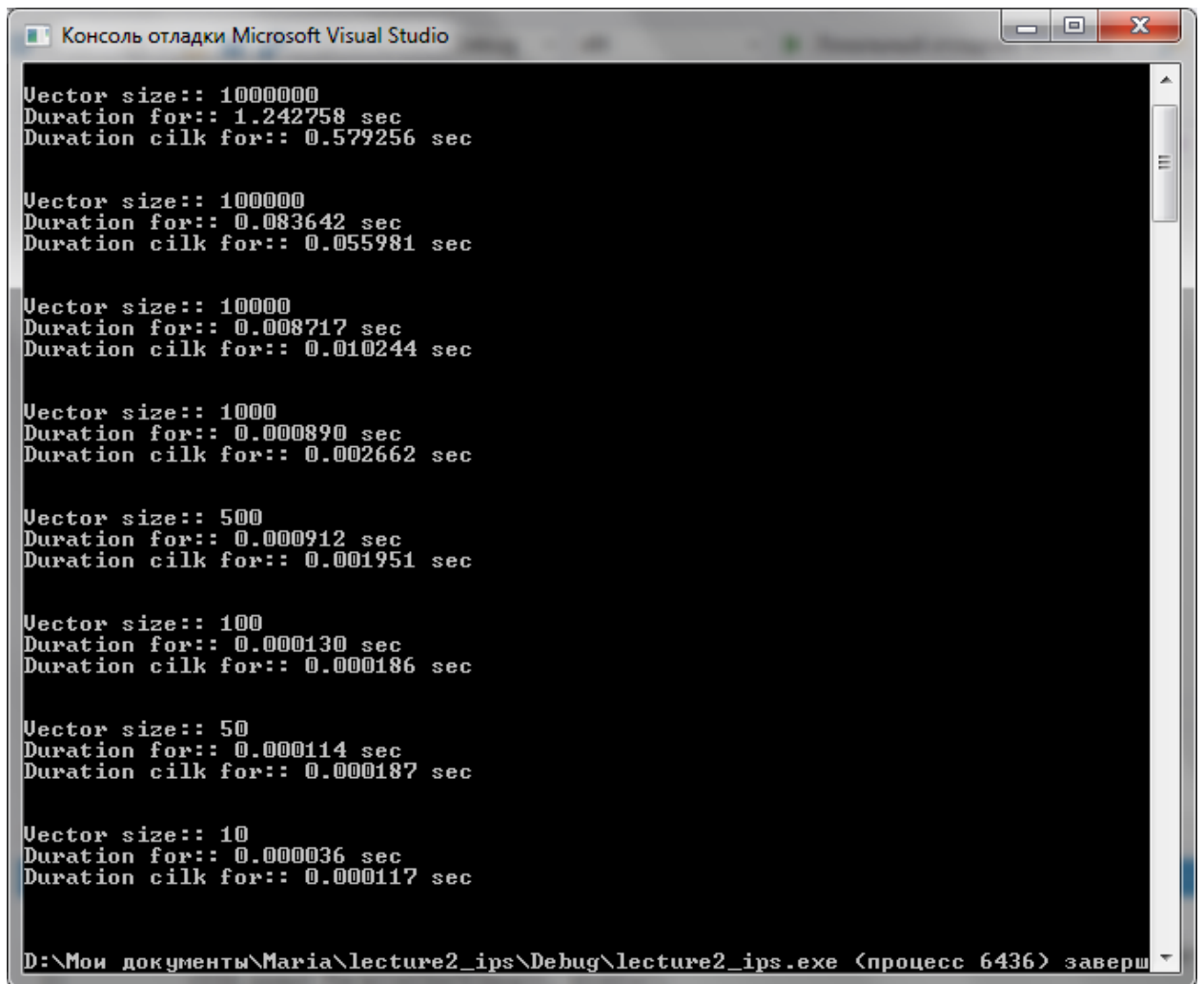
Maximal element = 25000 has index = 3707
Minimal element = 1 has index = 10371

After sorting:
Maximal element = 25000 has index = 999973
Minimal element = 1 has index = 0

Duration for sorting:: 1.609550 sec

D:\Мои документы\Maria\lecture2_ips\Debug\lecture2_ips.exe (процесс 8616) заверш
ает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу:
_
```

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***.



```
Консоль отладки Microsoft Visual Studio

Vector size:: 1000000
Duration for:: 1.242758 sec
Duration cilk for:: 0.579256 sec

Vector size:: 100000
Duration for:: 0.083642 sec
Duration cilk for:: 0.055981 sec

Vector size:: 10000
Duration for:: 0.008717 sec
Duration cilk for:: 0.010244 sec

Vector size:: 1000
Duration for:: 0.000890 sec
Duration cilk for:: 0.002662 sec

Vector size:: 500
Duration for:: 0.000912 sec
Duration cilk for:: 0.001951 sec

Vector size:: 100
Duration for:: 0.000130 sec
Duration cilk for:: 0.000186 sec

Vector size:: 50
Duration for:: 0.000114 sec
Duration cilk for:: 0.000187 sec

Vector size:: 10
Duration for:: 0.000036 sec
Duration cilk for:: 0.000117 sec

D:\Мои документы\Maria\lecture2_ips\Debug\lecture2_ips.exe (процесс 6436) заверш
```

5. Ответьте на вопросы:

Почему при небольших значениях *sz* цикл *cilk\_for* уступает циклу *for* в быстродействии?

Для *cilk\_for* в процессе компиляции тело цикла конвертируется в функцию, которая вызывается рекурсивно в соответствии со стратегией «разделяй и властвуй». Фактически множество итераций делится пополам до тех пор, пока после очередного деления количество итераций не будет превышать значения *grainsize* (максимальное количество итераций, на которое расщепляется множество итераций). Поэтому при малых значениях *sz* некоторое время тратится на разбиение тела цикла и планировку нагрузки между обработчиками (*worker*).

В каких случаях целесообразно использовать цикл *cilk\_for* ?

Если первое и последнее значения счетчика в цикле имеют большую разницу.

В чем принципиальное отличие параллелизации с использованием *cilk\_for* от параллелизации с использованием *cilk\_spawn* в паре с *cilk\_sync*?

*cilk\_spawn* – это конструкция, которая может быть использована непосредственно перед вызовом функции, чтобы указать системе, что данная функция (дочерняя) может выполняться параллельно с вызывающей (родительская). В ситуации, когда дальнейшие вычисления в родительской функции невозможны без результатов дочерней, необходимо использовать конструкцию синхронизации *cilk\_sync*.

Для `cilk_for` необходимо, чтоб итерации цикла были независимы по данным. Такое требование накладывается, чтобы каждая итерация могла быть выполнена в параллели с любой другой итерацией. При этом в процессе компиляции тело цикла конвертируется в функцию, которая вызывается рекурсивно в соответствии со стратегией «разделяй и властвуй», и каждый потомок всегда выполняет только по половине работы.