**Program**

#include <time.h> // clock\_t, clock, CLOCKS\_PER\_SEC

#include <random> // mt19937 and uniform\_int\_distribution

#include <iostream> // cout

#include <string> // string

#ifdef \_WIN32

#include <Windows.h>

#else

#include <unistd.h>

#endif

using namespace std;

// Generate sorted n size array with uniform increment

void generateSortIntArr(int\* arr, const int size, const int min, const int max)

{

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

arr[i] = i;

}

double sec(void)

{

return double(clock()) / double(CLOCKS\_PER\_SEC);

}

// iterative lambda to return arr[i], where arr[i] = x

// :param: int arr[], int size, int x

int bs\_it(int arr[], const int size, const int x)

{

int mid, low = 0, high = size - 1;

while (low <= high)

{

mid = (low + high) / 2;

if (arr[mid] == x)

return arr[mid];

else if (arr[mid] > x)

high = mid - 1;

else

low = mid + 1;

}

return -1;

}

// recursive lambda to return int where arr[i] = x

// :param: int arr[], int size, int x

int bs\_re(int arr[], const int size, const int x)

{

// inner recursive lambda

auto bs\_re\_imp = [](int arr[], const int low, const int high, const int x, auto& bs\_ref) mutable -> int

{

int mid = (low + high) / 2;

if (low > high)

return -1;

else if (arr[mid] == x)

return arr[mid];

else if (arr[mid] > x)

return bs\_ref(arr, low, mid - 1, x, bs\_ref);

else

return bs\_ref(arr, mid + 1, high, x, bs\_ref);

};

return bs\_re\_imp(arr, 0, size, x, bs\_re\_imp);

}

int main()

{

// setting up parameters

const double K = 1000;

const int n = 100000;

int arr[n];

generateSortIntArr(arr, n, 0, n);

// Iterative version

double T1 = sec();

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

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

if (bs\_it(arr, n, i) != i)

cout << "\nERROR: ";

double T2 = sec();

cout << "Total iterative run time = " << T2 - T1 << " seconds\n";

cout << "Individual iterative version run time = " << (((T2 - T1) / n) \* 1.E9) / K << " nanosec\n\n";

// Recursive version

T1 = sec();

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

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

if (bs\_re(arr, n, i) != i)

cout << "\nERROR";

T2 = sec();

cout << "Total recursive run time = " << T2 - T1 << " seconds\n";

cout << "Individual recursive version run time = " << (((T2 - T1) / n) \* 1.E9) / K << " nanosec\n\n";

// infinite loop to prevent program from closing before checking benchmark

cout << "On infinite loop to prevent software closing before recording benchmark.\n";

while (1)

{

cout << "Sleeping" << endl;

Sleep(10000);

}

return 0;

}

**Discussion**

**Tools:**

1. Microsoft’s Visual Studio C++ compiler version 16.7.4
2. Ubuntu on Window G++ version 7.5.0
3. Laptop with and without AC power plugged in.

**Results:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | Iterative | | Recursive | |
| **Charging** | Visual  C++ | Debug | 102.68 | 102.54 | 326.96 | 325.05 |
| Release | 37.09 | 37.3 | 37.27 | 37.17 |
| Linux  G++ | O0 | 90.625 | 90.3125 | 114.844 | 114.531 |
| O1 | 42.1875 | 42.1875 | 32.5 | 32.3437 |
| O2 | 40.625 | 40.4688 | 36.25 | 36.25 |
| O3 | 31.0938 | 31.0938 | 36.5625 | 36.5625 |
| Ofast | 31.0938 | 31.25 | 38.125 | 38.125 |
| **Battery** | Visual  C++ | Debug | 200.48 | 178.6 | 519.55 | 477.46 |
| Release | 93.4 | 75.91 | 78.44 | 74.17 |
| Linux  G++ | O0 | 118.906 | 117.5 | 140 | 139.531 |
| O1 | 50 | 50.625 | 39.0625 | 39.6875 |
| O2 | 47.1875 | 47.9688 | 42.6562 | 42.5 |
| O3 | 36.4062 | 37.0312 | 42.6562 | 43.75 |
| Ofast | 36.0938 | 37.3438 | 45.9375 | 45.3125 |

**Discussion:**

1. Processors are under clocked when laptop run on battery, plausibly a strategy implemented by laptop manufacturer to prolong battery lifetime.
2. Iteration function is more efficient than recursive function most of the time.
3. Some version of the software has recursive function running more efficiently, including: Visual Studio release version (battery ), G++ O1, O2 version (both in battery charging).
   1. It might be tail-call optimization by the compiler. It appears that tail-call optimization might make recursive function better than iterative function.
   2. Upon further discussion with Professor, no conclusion was given for this behavior.
4. Release version of the Visual C++ is optimized by the compiler.
5. O1 version of the G++ has the shortest execution time for unknown reason, given that O2, O3, and Ofast should be more optimized for execution time according to [this table](https://www.rapidtables.com/code/linux/gcc/gcc-o.html).
6. G++ is able to run more efficiently than Visual.
7. I observed a consistent acceleration in execution time when using Visual C++ compiler. Perhaps it is due to caching.