Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Сортировка Шелла с простым слиянием»**

**Выполнил**:

студент группы 381708-2

Фамилия И.О.Утшо М.М.

**Проверил**:

Валентин Волокитин

Нижний Новгород

2020

**Содержание**

[Введение](#_Toc26446451) 3

[Постановка задачи](#_Toc26446451) 4

[Метод решения](#_Toc26446452) 5

[Описание алгоритма](#_Toc26446453) 14

Схема распараллеливания 15

Реализации с исползованием OpenMP15

Реализации с исползованием TBB  [16](#_Toc26446454)

[Результаты экспериментов 8](#_Toc26446456)

[Заключение](#_Toc26446457) 18

[Литераура](#_Toc26446458) 19

# Введение

Shell sort-это алгоритм сортировки, разработанный Дональдом Шеллом в 1959 году, который является обобщением insertion sort, который использует тот факт, что insertion sort эффективно работает на входных данных, которые уже почти отсортированы.

Он улучшает сортировку вставки, позволяя сравнивать и обмениваться элементами, которые находятся далеко друг от друга. Другими словами, он улучшает сортировку пузырьков и сортировку вставки, перемещая элементы из порядка более чем на одну позицию одновременно. Последний шаг сортировки оболочки-это простая сортировка вставки, но к тому времени массив данных гарантированно будет почти отсортирован.

# Постановка задачи

В рамках данных лабораторных работ составляется задача Сортировка Шелла с простым слиянием

Для решения данной задачи необходимо выполнить следующие пункты

1. Изучать алгоритм сортировка шелла
2. Написать параллельные реализатции shell sequence, OpenMP , TBB

# Метод решения

Sequence shell

// shellsortsequence.cpp : This file contains the 'main' function. Program execution begins and ends there.

//COPYRIGHT UTSHO\_MOMEN

#include "pch.h"

#include <iostream>

using namespace std;

//a[] is sort , n is size

void ShellSort(int a[], int n)

{

int gap;

int j;

int i;

int element;

for (gap = n / 2; gap > 0; gap /= 2)

{

for (i = gap; i < n; i++)

{

element = a[i];

for (j = i; j >= gap; j -= gap)

{

if (element < a[j - gap])

a[j] = a[j - gap];

else

break;

}

a[j] = element;

}

}

}

void print(int a[], int size)

{

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

cout << a[i] <<"";

cout << endl;

}

int main()

{

int n;

int a[20];

cout << "enter element:" << endl;

cin >> n;

cout << "enter elements the one that you put earlier:" << endl;

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

{

cin >> a[i];

}

cout << "array seq before sorting: ";

print(a, n);

ShellSort(a, n);

cout << "array seq after sorting: ";

print(a, n);

}

OpenMP Shell

#include "pch.h"

#include <iostream>

#include "malloc.h"

#include "omp.h"

#include <time.h>

#include <ctime>

#include <cstdlib>

#include <cstring>

#define MAX\_SIZE 1000

using namespace std;

void printarray(int \*array, int length) //print array elements

{

int i = 0;

for (i = 0; i < length; i++)

printf("%d\n", array[i]);

}

void shellsort(int numbers[], int array\_size)

{

{

int gap;

int j;

int i;

int element;

for (gap = array\_size / 2; gap > 0; gap /= 2)

{

#pragma omp parallel for shared( numbers, gap,array\_size) private(i,j,element) default(none)

for (i = gap; i < array\_size; i++)

{

#pragma omp parallel

element = numbers[i];

for (j = i; j >= gap; j -= gap)

{

#pragma omp critical

if (element < numbers[j - gap])

numbers[j] = numbers[j - gap];

}

numbers[j] = element;

}

}

}

}

void merge(int a[], int array\_size, int t[]) {

int i = 0;

int j = array\_size / 2;

int ti = 0;

while (i < array\_size / 2 && j < array\_size) {

if (a[i] < a[j]) {

t[ti] = a[i];

ti++; i++;

}

else {

t[ti] = a[j];

ti++; j++;

}

}

while (i < array\_size / 2)

{

t[ti] = a[i];

ti++; i++;

}

while (j < array\_size)

{

t[ti] = a[j];

ti++; j++;

}

memcpy(a, t, array\_size \* sizeof(int));

}

void mergeSort(int \*numbers, int t[], int array\_size)

{

if (numbers > 0) {

for (int i = 0; i < array\_size / 2; i++) {

t[i] = numbers[i];

}

mergeSort(numbers, t, array\_size / 2);

for (int i = array\_size / 2; i < array\_size; i++) {

t[i] = numbers[i];

}

merge(numbers, array\_size + 1, 0);

}

};

int main() {

int threads = 2;

int t[MAX\_SIZE];

int array\_size = 2000;

int numbers[2000];

int i;

srand((time(NULL)));

#pragma omp parallel for num\_threads(i)

for (i = 0; i < array\_size; i++)

{

numbers[i] = (rand() % 100000) + 1;

}

double startTime = clock();

cout << "enter element:" << endl;

cin >> array\_size;

cout << "enter elements the one that you put earlier:" << endl;

for (int i = 0; i < array\_size; i++)

{

cin >> numbers[i];

}

cout << "array seq before sorting: ";

printarray(numbers, array\_size);

shellsort(numbers, array\_size);

cout << "array seq after sorting: ";

printarray(numbers, array\_size);

//cout << "\tSorted Array Elements with shell" << endl;

//for (int i = 0; i < array\_size; i++) {

// cout << numbers[i] << "\t";

//}

//shellsort(numbers, array\_size);

if (array\_size < 100) {

std::cout << "Sorted array:\t ";

printarray(numbers, array\_size);

}

shellsort(numbers, array\_size);

double endTime = clock();

int totalTime = endTime - startTime;

cout << "This is the time it took to run.\n" << endl;

cout << totalTime / threads << array\_size << endl;

}

TBB Shell

// tbbone.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

//clang++ -std=c++17 -g -DTBB\_USE\_DEBUG=1 tbbone.cpp -ltbb\_debug -ltbbmalloc\_debug -o tbbone.exe

//clang++ -std=c++17 tbbone.cpp -ltbb -ltbbmalloc

#include "pch.h"

#include <iostream>

#include "malloc.h"

#include <time.h>

#include <cstdlib>

#include <stdlib.h>

#include "tbb/tbb.h"

#include <thread>

#include <ctime>

#include <algorithm>

#include <utility>

using namespace tbb;

#define MAX\_SIZE 1000

//a[] is sort , n is size

void print(int a[], int n) {

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

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

std::cout << std::endl;

}

void merge(int a[], int n, int gap) {

for (int j = gap; j < n; j += gap) {

int key = a[j];

int i = j - gap;

while (i >= 0 && a[i] > key) {

a[i + gap] = a[i];

i -= gap;

}

a[i + gap] = key;

}

}

void shellsort(int a[], int n) {

for (int gap = n / 2; gap > 0; gap /= 2)

tbb::parallel\_for(tbb::blocked\_range<int>(0, gap),

[&](const tbb::blocked\_range<int> &r) {

for (int i = r.begin(); i < r.end(); i++) {

merge(a + i, n - i, gap);

}

});

}

int main() {

int n = 2000;

int a[2000];

const int threadNum = 2;

tbb::tick\_count startTime;

double maintime;

double defineTime;

double parallelTime;

srand((time(NULL)));

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

{

a[i] = (rand() % 100000) + 1;

}

std::cout << "enter elements :";

std::cin >> n;

std::cout << "enter elements the one that you put earlier:";

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

{

std::cin >> a[i];

}

std::cout << "array seq before sorting: ";

print(a, n);

startTime = tbb::tick\_count::now();

maintime = (tbb::tick\_count::now() - startTime).seconds();

tbb::task\_scheduler\_init init(1);

startTime = tbb::tick\_count::now();

shellsort(a, n);

defineTime = (tbb::tick\_count::now() - startTime).seconds();

init.terminate();

init.initialize(threadNum);

startTime = tbb::tick\_count::now();

shellsort(a, n);

parallelTime = (tbb::tick\_count::now() - startTime).seconds();

init.terminate();

if (n < 100) {

std::cout << "Sorted array: ";

print(a, n);

}

shellsort(a, n);

std::cout << "\nParallel shell sort " << std::endl;

std::cout << "maintimestarting: " << maintime << std::endl;

std::cout << "\ndefine time: " << defineTime << std::endl;

std::cout << "\nparallel time: " << parallelTime << std::endl;

std::cout << "\nparallel efficencty= ";

std::cout << defineTime / parallelTime << std::endl;

return 0;

}

# ОПИСАНИЕ АЛГОРИТМА

Сортировка оболочки-это алгоритм, который сначала сортирует элементы далеко друг от друга и последовательно уменьшает интервал между элементами, подлежащими сортировке. Это универсальная версия типа insert.

В сортировка Шелла, элементы сортируются в определенные промежутки времени. Интервал между элементами постепенно уменьшается в зависимости от используемой последовательности. Производительность сортировки оболочки зависит от типа последовательности, используемой для этого входного массива.

# СХЕМА РАСПАРАЛЛЕЛИВАНИЧ

1. OPEN MP

В приведенном ниже задании по программированию я попрошу вас написать программу на языке C++, реализующую другой параллельный алгоритм.

OpenMP - это система для написания параллельных программ на языке C++. OpenMP-это комбинация набора расширений компилятора и библиотеки OpenMP. Расширения компилятора позволяют реализовать функции параллельного программирования в вашей программе с помощью директив компилятора, в частности прагм.

Вот исходный код для версии параллельного алгоритма сортировки оболочки на языке C++.

Прагмы OpenMP, встроенные в этот код, довольно просты в использовании.

#pragma omp parallel sections

{

#pragma omp section

x = f(i);

#pragma omp section

y = f(i-1);

}

Комбинация прагмы параллельных секций и фигурных скобок устанавливает неявную синхронизацию, которая происходит после правой фигурной скобки.

Когда вы используете параллель Для или раздел для перемещения некоторого кода в другой поток, работающий на другом ядре, вы захотите использовать некоторую осторожность, чтобы убедиться, что вы отличаете переменные, которые являются общими для всех потоков, от переменных, предназначенных для использования исключительно в одном потоке. Вы делаете это различие, добавляя директивы общих и частных переменных в свои прагмы OpenMP. Если вы оглянетесь на приведенный выше код для сортировки параллельным слиянием, то увидите, что в каждом случае я тщательно указывал, какие переменные являются общими, а какие-частными.

2.tbb

Библиотека стандартных шаблонов C++ (STL) в основном включает в себя следующие операции:

применяется к последовательностям. Есть некоторые аутдайеры, такие как std: и std:

это может быть применено к значениям, но по большей части к алгоритмам, таким как

с std: : for\_each, с std: : найти, СТД: преобразование, СТД: копирование и

std:: сортировка, применяются к последовательностям элементов. Этот фокус на последовательностях является

удобно, когда мы хотим работать с контейнерами, поддерживающими итераторы, но

может быть несколько громоздким, если мы хотим выразить то, чего нет

работайте на контейнерах. Позже в этой главе мы увидим, что иногда мы можем

"думайте нестандартно" и используйте пользовательские итераторы, чтобы заставить некоторые алгоритмы действовать

больше похоже на общие петли.

Объяснение того, что делает каждый алгоритм STL, выходит за рамки этой главы

и книга. Есть целые книги, написанные о стандартном шаблоне C++

Библиотека и как ее использовать, включая стандартную библиотеку C++: учебник и

Ссылка Николая хосуттиса (Addison-Wesley Professional). В этой главе мы только

сосредоточьтесь на том, что означают для них политики выполнения, впервые введенные в C++17

алгоритмы и то, как их можно использовать вместе с ТББ.

Большинство алгоритмов STL, указанных в стандарте C++ hase, перегружаются в

C++17, которые принимают политику выполнения. Кроме того, было разработано несколько алгоритмов ner

добавлено потому, что они особенно полезны в параллельных программах или потому, что

комитет хотел избежать изменений в семантике. Мы можем найти именно то, что

алгоритмы поддерживают политику выполнения, просматривая сам стандарт или онлайн

на веб-сайтах, таких как http: //en. cppreference, com / w/cpp / алгоритм.

Как получить и использовать копию Parallel STL, использующую TBB

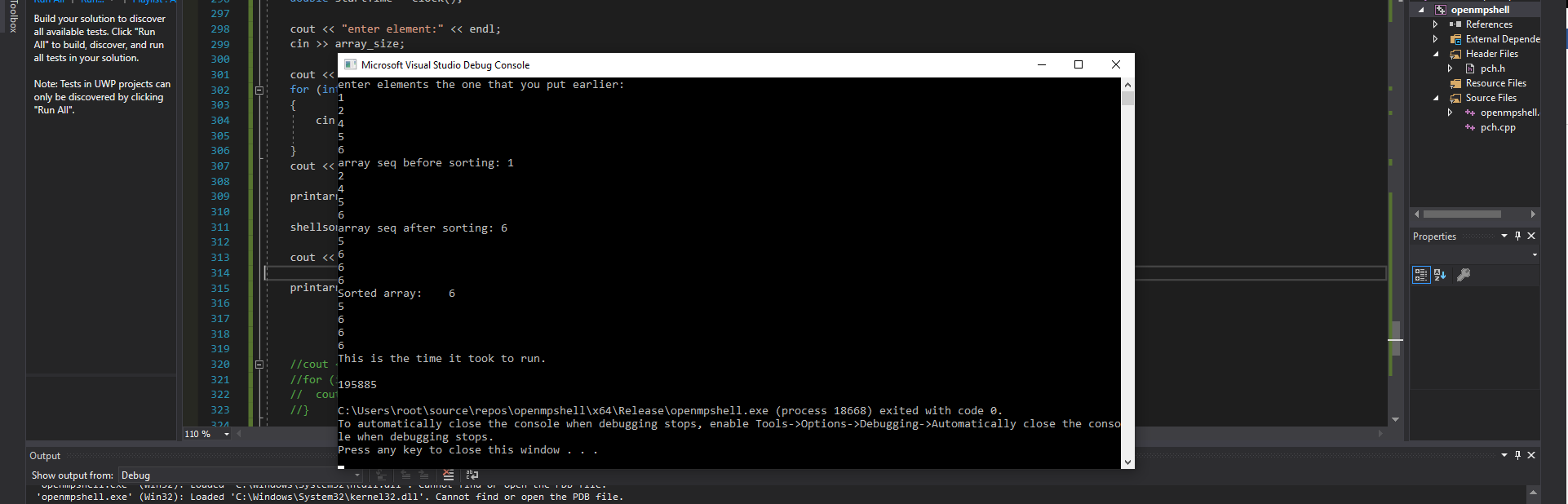
Подробные инструкции по загрузке и установке параллельного STL от Intel приведены ниже

приведено в Главе 1 в разделе " Получение Резьбонарезных строительных блоков

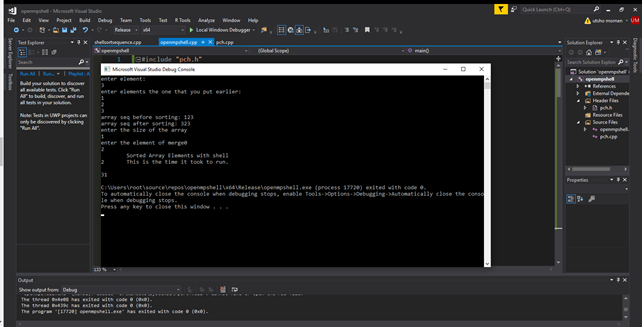
Зорге-3

# Результаты экспериментов

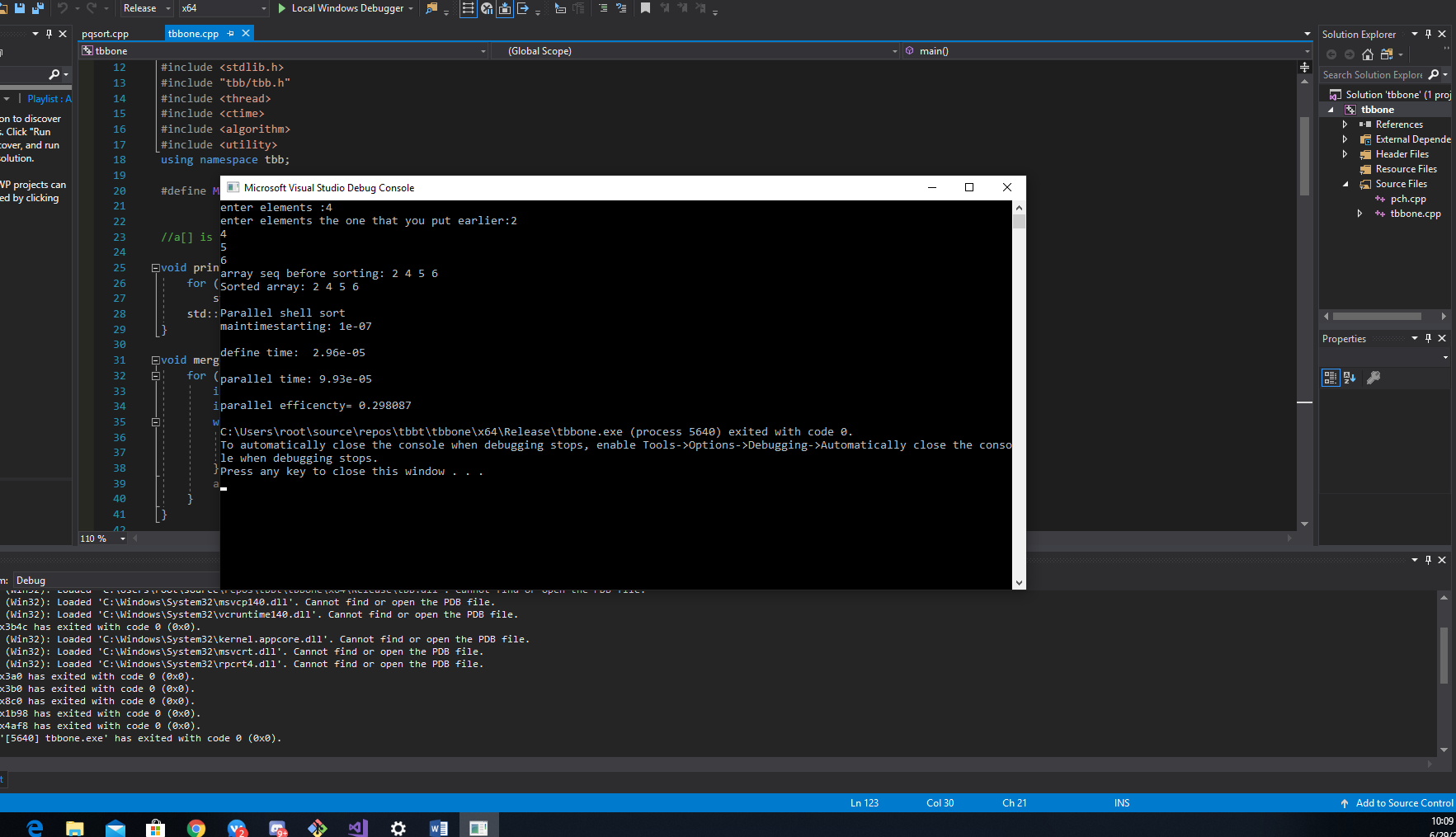
Sequence shell sort



openmp



Tbb



# Заключение

Shell sort - это высокоэффективный алгоритм, который приходит к улучшению по сравнению с сортировкой вставки.

В то время как сортировка вставки работает путем увеличения ее элементов на 1, shell sort использует параметр “gap” для разделения массива на суб-массивы, элементы которых разделены “gap”. Затем мы можем отсортировать отдельный список с помощью сортировки вставки, чтобы получить полный отсортированный массив.

Сортировка оболочки выполняется быстрее, чем сортировка вставки, и занимает меньше времени для сортировки массива по сравнению с сортировкой вставки. В нашем предстоящем уроке мы рассмотрим все о технике сортировки кучи для сортировки структур данных.

1.sequence

В процессе выполнения данных лабораторных работ была реализована после-довательная версия сортировка Шелаа

2.OpenMp,TBB

Алгоритм имеет среднюю временную сложность сортировки оболочки и наихудшую временную сложность создания сортировки оболочки

это один из самых быстрых доступных алгоритмов последовательной сортировки.

Поскольку мы не стремимся создать самый быстрый алгоритм параллельной сортировки, а просто пытаемся показать некоторые проблемы

и решения с OpenMP,TBB мы выбрали это легко и широко

используемый алгоритм в качестве основы для наших экспериментов, а не

более продвинутый и сложный (последовательный или параллельный).

# ЛИТЕРАТУРА

1.A K Verma

2.A New Approach for Sorting List to Reduce Execution Time, volume 7

Posted: 2013-10-13

3.A K S Shukla

Review of Radix Sort & Proposed Modified Radix Sort for Heterogeneous Data Set in 4.Distributed Computing Environment, volume 2

Posted: 2012

5.J Hammad

A Comparative Study between Various Sorting Algorithms, volume 15

Posted: 2015-03

6.S K P Megha Jain

Bitonic Sorting Algorithm: A Review, volume 113

Posted: 2015-03