Lab 6

Lưu ý:

- Những bài làm giống nhau sẽ bị 0 điểm
- Với những bài lập trình, cần phải copy mã nguồn và chụp màn hình kết quả, đưa vào file word
- Địa chỉ nộp bài: ctdlgt.bku@gmail.com
- Hạn chót nộp bài: 23 giờ ngày 17/8/2023

Nội dung:

- Hoàn thiện lớp Heap
- Hiện thực các giải thuật sắp xếp đã học
- Đo thời gian chạy thực tế đối với các giải thuật sắp xếp

I. Sinh viên đọc hiểu mã nguồn sau:

```
#include <iostream>
#include <time.h>
#include <windows.h>
#include <math.h>

using namespace std;

#define             ARRAY_SIZE    50000

#define             INC_SORT      0
#define             DEC_SORT      1

#define             MAX_HEAP      0
#define             MIN_HEAP      1

class Heap
{
public:
	int*   arr;
	int           count;
	int           capacity;
	int           heap_type;
public:
	int    getCount() { return count;}
	void CopyData(int* data, int size)
	{
		memcpy(this->arr, data, sizeof(int)*size);
		this->count = size;
	}
	void CreateHeap(int capacity, int heap_type)
	{
		this->heap_type = heap_type;
		this->count = 0;
```

```
		this->capacity = capacity;
		this->arr = new int[this->capacity];
		if(this->arr == NULL)
		{
			cout << "Not enough memory";
			exit(-1);
		}
	}
	~Heap()
	{
		delete [] this->arr;
	}
	void ReheapUp(long position)
	{
		if(position > 0)
		{
			long parent = (position - 1)/2;

			// For max-heap
			if(this->heap_type == MAX_HEAP && this->arr[position] > this-
>arr[parent])
			{
				int temp = this->arr[position];
				this->arr[position] = this->arr[parent];
				this->arr[parent] = temp;
				ReheapUp(parent);
			}
		}
	}
	void ReheapDown(int position, int lastPosition)
	{
		long leftChild = 2*position + 1;
		long rightChild = 2*position + 2;
		long child;

		//For max-heap
		if(this->heap_type == MAX_HEAP)
		{
			if(leftChild <= lastPosition)
			{
				if(rightChild <= lastPosition && this->arr[rightChild] > this-
>arr[leftChild])
					child = rightChild;
				else
					child = leftChild;

				if(this->arr[child] > this->arr[position])
				{
					int temp = this->arr[child];
					this->arr[child] = this->arr[position];
					this->arr[position] = temp;
					ReheapDown(child, lastPosition);
				}
			}
		}
	}
```

```
	bool InsertHeap(int DataIn)
	{
		if(this->count == this->capacity)
			return false;
		else
		{
			this->arr[this->count]= DataIn;
			ReheapUp(this->count);
			this->count++;
			return true;
		}
	}
	bool DeleteHeap(int &DataOut)
	{
		if(this->count <= 0)
			return false;
		else
		{
			DataOut = this->arr[0];
			this->arr[0] = this->arr[count - 1];
			count = count - 1;
			ReheapDown(0, count-1);
			return true;
		}
	}
	void BuildHeap()
	{
		long position = this->count/2 - 1;

		while(position >= 0)
		{
			ReheapDown(position, count - 1);
			position--;
		}

	}
	bool IsHeap()
	{
		long position = this->count/2 - 1;
		long lastPosition = this->count - 1;

		while(position >= 0)
		{
			long leftChild = 2*position + 1;
			long rightChild = 2*position + 2;
			long child;

			//For max-heap
			if(this->heap_type == MAX_HEAP)
			{
				if(leftChild <= lastPosition)
				{
					if(rightChild <= lastPosition && this-
>arr[rightChild] > this->arr[leftChild])
						child = rightChild;
					else
```

```
                                        child = leftChild;

                                if(this->arr[child] > this->arr[position])
                                        return false;
                                }
                        }

                        position--;
                }
                return true;
        }
        void PrintHeap()
        {
                for(long i=0; i<this->count; i++)
                        cout << this->arr[i] << " ";
                cout << endl;
        }
};
int*    arr1=NULL;
Heap    heap;

long lrand()
{
#define         CHAR_BITS     8
        long r = 0;
        for (int i = 0; i < sizeof(long)/sizeof(int); i++)
        {
                r = r << (sizeof(int) * CHAR_BITS);
                r |= rand();
        }
        return r;
}

bool IsSorted(int nType)
{
        if(nType == DEC_SORT)
                for(int i = 0; i< ARRAY_SIZE; i++)
                        if(arr1[i] != ARRAY_SIZE - 1 - i)
                                return false;
        else if(nType == INC_SORT)
                for(int i = 0; i< ARRAY_SIZE; i++)
                        if(arr1[i] != i)
                                return false;

        return true;
}

void ShuffleData(int *arr)
{
        // Shuffle data
        long i;
        srand(time(NULL));
        for(i = ARRAY_SIZE - 1; i>0 ; i--)
        {
                long j = lrand() % ARRAY_SIZE;
```

```
			int	temp = arr[i];
			arr[i] = arr[j];
			arr[j] = temp;
		}
}

void InsertionSort(int* arr)
{
		long count = ARRAY_SIZE, current;

		current = 1;

		while(current < count)
		{
			long temp = arr[current];
			long walker = current - 1;

			while(walker >= 0 && temp < arr[walker])
			{
				arr[walker + 1] = arr[walker];
				walker--;
			}

			arr[walker+1] = temp;
			current++;
		}
}


void Sorting(void (*SortFunc)(int*), int nSortType = INC_SORT)
{
		DWORD startTime;
		DWORD endTime;
		long i;
		char	strMessage[80];

		if(nSortType == INC_SORT)
			strcpy(strMessage, "The array is sorted in increasing order");
		else
			strcpy(strMessage, "The array is sorted in decreasing order");
		//Sorting random array
		cout << "Sorting an random array ... " << endl;

		startTime = GetTickCount();
		(*SortFunc)(arr1);
		endTime = GetTickCount();

		if(IsSorted(nSortType))
			cout << strMessage << endl;
		else
			cout << "The array is unorder" << endl;

		cout << "Time (ms) :  "<< (endTime - startTime)/1000.0 << endl;

		//Sorting increasing array
		for(i = 0; i<ARRAY_SIZE; i++)
```

```
		arr1[i] = i;

	cout << "Sorting an increasing array ... " << endl;

	startTime = GetTickCount();
	(*SortFunc)(arr1);
	endTime = GetTickCount();

	if(IsSorted(nSortType))
		cout << strMessage << endl;
	else
		cout << "The array is unorder" << endl;

	cout << "Time (ms) :  "<< (endTime - startTime)/1000.0 << endl;

	//Sorting decreasing array
	for(i = 0; i<ARRAY_SIZE; i++)
		arr1[i] = ARRAY_SIZE - 1 - i;

	cout << "Sorting an decreasing array ... " << endl;

	startTime = GetTickCount();
	(*SortFunc)(arr1);
	endTime = GetTickCount();

	if(IsSorted(nSortType))
		cout << strMessage << endl;
	else
		cout << "The array is unorder" << endl;

	cout << "Time (ms) :  "<< (endTime - startTime)/1000.0 << endl;
}

int main()
{
	//Create array;
	arr1 = new int[ARRAY_SIZE];

	if(arr1 == NULL)
	{
		cout << "Not enough memory";
		return 0;
	}
	for(long i = 0; i<ARRAY_SIZE; i++)
		arr1[i] = i;

	heap.CreateHeap(ARRAY_SIZE, MIN_HEAP);

	cout << "ARRAY SIZE: "<< ARRAY_SIZE << endl << endl;

	cout << "================================================"<<endl;
	cout << "                    INSERTION SORT" << endl;
	cout << "================================================"<<endl;
	ShuffleData(arr1);
	Sorting(InsertionSort);
```

```
        //Release memory
        delete [] arr1;

        return 0;
}
```

-Lớp Heap đã hiện thực cho trường hợp max-heap
-Mảng arr1 chứa dữ liệu từ 0 đến ARRAY_SIZE-1
-Hàm ShuffleData(…) trộn dữ liệu
-Hàm IsSorted(…) kiểm tra xem dữ liệu đã được sắp xếp hay chưa
-Hàm Sorting(…) nhận đối số là một hàm. Trong đoạn chương trình trên hàm đó là InsertionSort. Trong hàm Sorting(…), lần lượt tiến hành sắp xếp một dãy có thứ tự ngẫu nhiên, một dãy có thứ tự tăng dần và một dãy có thứ tự giảm dần, đồng thời tiến hành đo đạc thời gian tương ứng.

II. Sinh viên thực hiện các công việc sau:

- Hoàn thiện lớp Heap, để lớp này có thể tạo được min-heap
- Hiện thực các giải thuật sắp xếp sau:
    - Sắp xếp Shell, sắp xếp dãy số theo thứ tự tăng dần
    - Sắp xếp chọn trực tiếp, sắp xếp dãy số theo thứ tự tăng dần
    - Sắp xếp Heap, **sắp xếp dãy số theo thứ tự giảm dần**
    - Sắp xếp nổi bọt, sắp xếp dãy số theo thứ tự tăng dần
    - Sắp xếp QuickSort, sắp xếp dãy số theo thứ tự tăng dần
    - Sắp xếp MergeSort, sắp xếp dãy số theo thứ tự tăng dần
- Tiến hành đo đạc thời gian với các giá trị khác nhau của ARRAY_SIZE (ít nhất 10 giá trị). Với mỗi phương pháp sắp xếp, ghi kết quả vào bảng có dạng như sau.

ARRAY_SIZE	Insertion Sort		
	Dãy ngẫu nhiên (s)	Dãy tăng dần (s)	Dãy giảm dần (s)

- Có nhận xét gì về kết quả đo đạc thực tế với giá trị độ phức tạp tính theo lý thuyết