

**Tên nhóm: N04**

**Lớp: CS523.L21**

**Thành viên 1: Vũ Nguyễn Nhật Thanh – 19522246**

**Thành viên 2: Nguyễn Phi Long – 19521791**

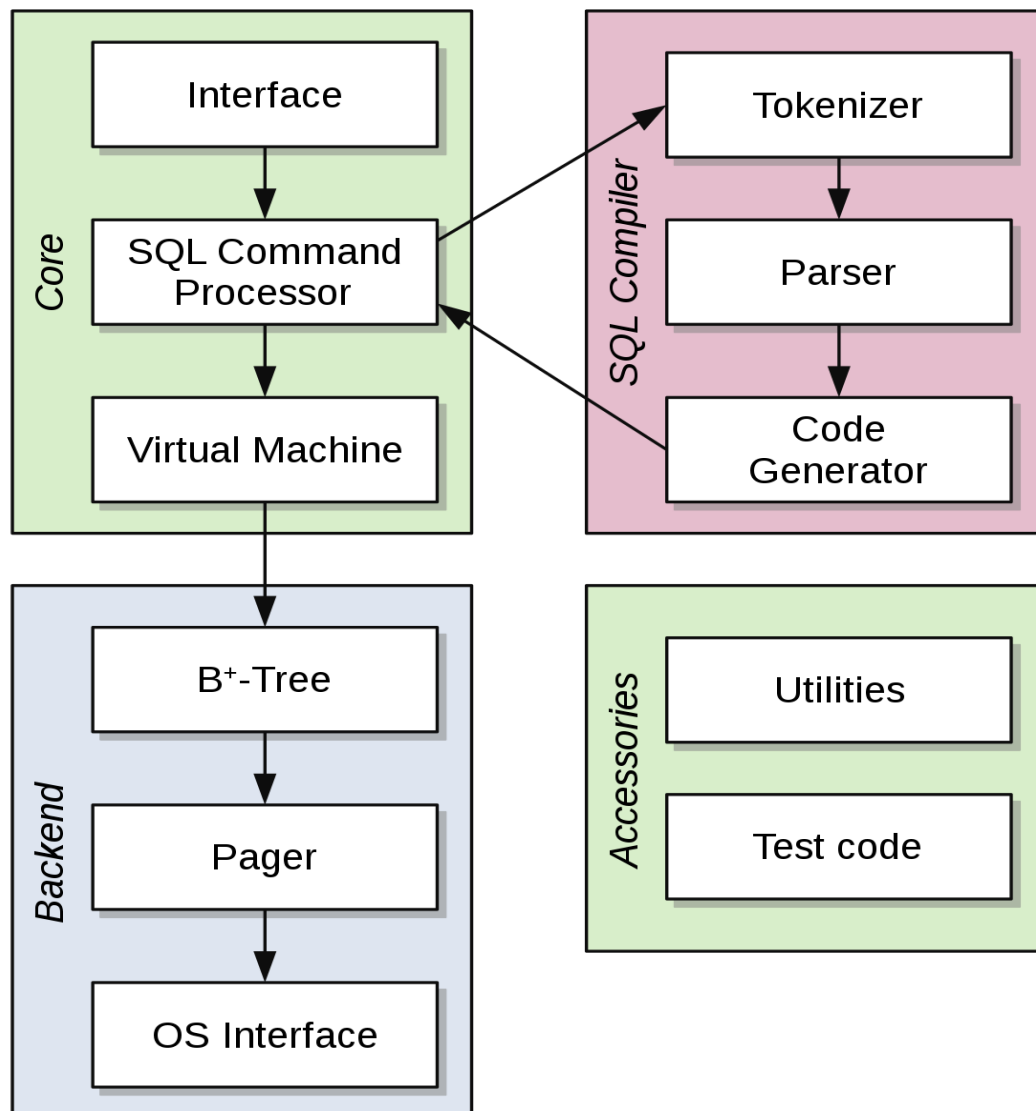
# **Báo Cáo Seminar B-tree**

## **I. SQLite**

### **1.1) Đặc điểmSQLite**

- SQLite là hệ thống cơ sở dữ liệu nhỏ gọn, hoàn chỉnh, có thể cài đặt bên trong các trình ứng dụng khác. SQLite được Richard Hipp viết dưới dạng thư viện bằng ngôn ngữ lập trình C.
- Có thể tương tác bằng C++, Java, C#, Python, Perl, Ruby, Visual Basic, and Tcl.
- Mã nguồn mở.
- Yêu cầu bộ nhớ không đến 300 KiB nhưng lại có hiệu năng cao hoạt động offline, không cần sever.
- Các công ty sau đây đã áp dụng SQLite cho các sản phẩm của mình:
  - Adobe: dùng SQLite cho chương trình Adobe Lightroom
  - Apple: dùng SQLite cho Apple Mail, Safari
  - McAfee: dùng SQLite trong phần mềm diệt virus

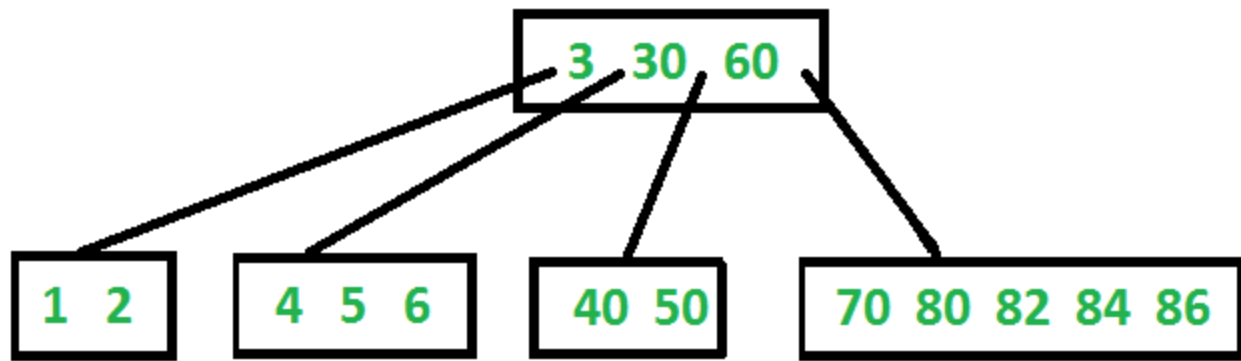
## 1.2) Cấu trúc của SQL



## II. B-tree, B+tree

### 1) Thuộc tính của B-tree

- Btree là cây tìm kiếm tự cân bằng
- Tất cả các lá đều ở cùng cấp
- Root có ít nhất 2 con
- Mỗi node trừ Root có thể có tối đa  $m$  con và ít nhất  $m/2$  con.
- Mỗi node có thể chứa tối đa  $m - 1$  khóa và ít nhất  $m/2 - 1$  khóa.



## 2) Bậc của Btree là gì

- Degree đại diện cho giới hạn dưới về số node con mà Btree có thể có (trừ Root).
- Order đại diện cho giới hạn trên về số node con mà Btree có thể có.
- Độ sâu của cây chỉ có thể tăng khi thực hiện chia Root node, mỗi node lá có cùng độ sâu và gần như tương đương với số cặp khóa/value, Btree duy trì được sự cân bằng và tìm kiếm nhanh

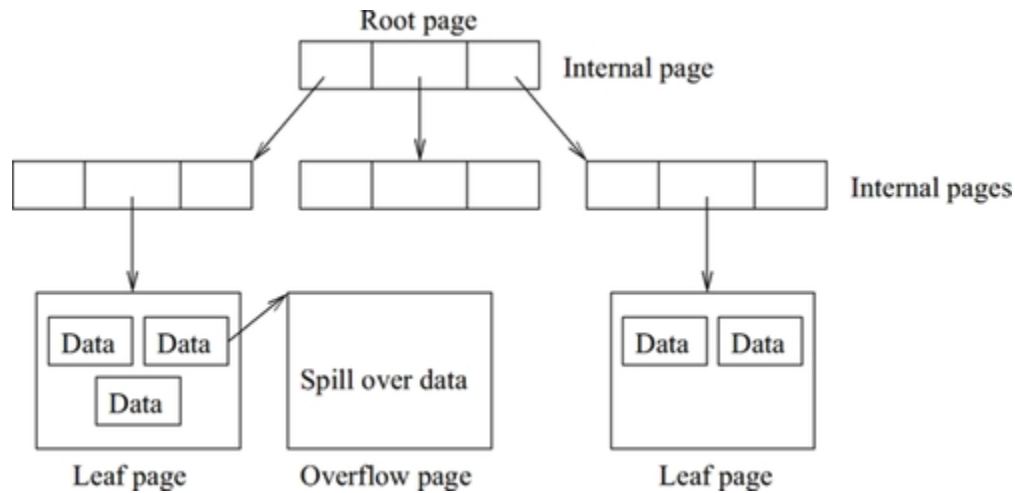
## 3) Tại sao B-tree được sử dụng trong database?

- Một database có thể thực hiện tìm kiếm nhị phân bằng việc sử dụng index hoặc tìm kiếm tuần tự bằng việc lướt qua toàn bộ các phần tử, nếu không đánh chỉ mục, database sẽ đọc từng record để tìm ra record mong muốn, mỗi node trong B-tree và B+tree được lưu trữ trong các Page.
- Btree được sử dụng trong database do nó có độ phức tạp ổn định, và nó được tối ưu cho việc đọc ghi trên đĩa cứng.

	Unsorted Array of rows	Sorted Array of rows	Tree of nodes
Pages contain	only data	only data	metadata, primary keys, and data
Rows per page	more	more	fewer
Insertion	$O(1)$	$O(n)$	$O(\log(n))$
Deletion	$O(n)$	$O(n)$	$O(\log(n))$
Lookup by id	$O(n)$	$O(\log(n))$	$O(\log(n))$

#### 4) B+tree

- B+tree là cây có cấu trúc mà mỗi node sẽ tương ứng với 1 disk block và có các thuộc tính sau:
  - + Là cây cân bằng
  - + Các node lá có cùng độ sâu.
  - + Một internal node chứa một list khóa và một list pointer.
  - + Các node (trừ Root) luôn đầy một nửa.



- Sự khác nhau giữa internal node và node lá:

For an order-m tree...	Internal Node	Leaf Node
Stores	keys and pointers to children	keys and values
Number of keys	up to m-1	as many as will fit
Number of pointers	number of keys + 1	none
Number of values	none	number of keys
Key purpose	used for routing	paired with value
Stores values?	No	Yes

- Sự khác nhau giữa Btree và B+tree:

	B-tree	B+ tree
Pronounced	"Bee Tree"	"Bee Plus Tree"
Used to store	Indexes	Tables
Internal nodes store keys	Yes	Yes
Internal nodes store values	Yes	No
Number of children per node	Less	More
Internal nodes vs. leaf nodes	Same structure	Different structure

### III. Thực nghiệm

#### 1) Cấu hình máy test

System:

Kernel: 5.12.3-zen1-1-zen x86\_64 bits: 64

Desktop: GNOME 40.1

Distro: Garuda Linux

Machine:

Type: Laptop

System: Dell

product: Inspiron 3580.

Memory:

RAM: total: 7.62 GiB used: 5.29 GiB (69.3%)

Array-1: capacity: 32 GiB slots: 2 EC: None

Device-1: DIMM A size: 4 GiB speed: spec: 2667 MT/s actual: 2400 MT/s

Device-2: DIMM B size: 4 GiB speed: spec: 2667 MT/s actual: 2400 MT/s

CPU:

Info: Quad Core

model: Intel Core i5-8265U

bits: 64

type: MT MCP

cache: L2: 6 MiB

Speed: 3573 MHz

min/max: 400/3900 MHz

Core speeds (MHz)

## Graphics:

Device-1: Intel UHD Graphics 620 driver: i915 v: kernel

Device-2: AMD Jet PRO [Radeon R5 M230 / R7 M260DX / Radeon 520 Mobile]

driver: radeon v: kernel

Device-3: Microdia Integrated\_Webcam\_HD type: USB driver: uvcvideo

Display: server: X.Org 1.20.11 driver: loaded: ati,intel,radeon unloaded: modesetting

resolution: 1920x1080~60Hz

OpenGL: renderer: Mesa Intel UHD Graphics 620 (WHL GT2) v: 4.6 Mesa 21.1.0

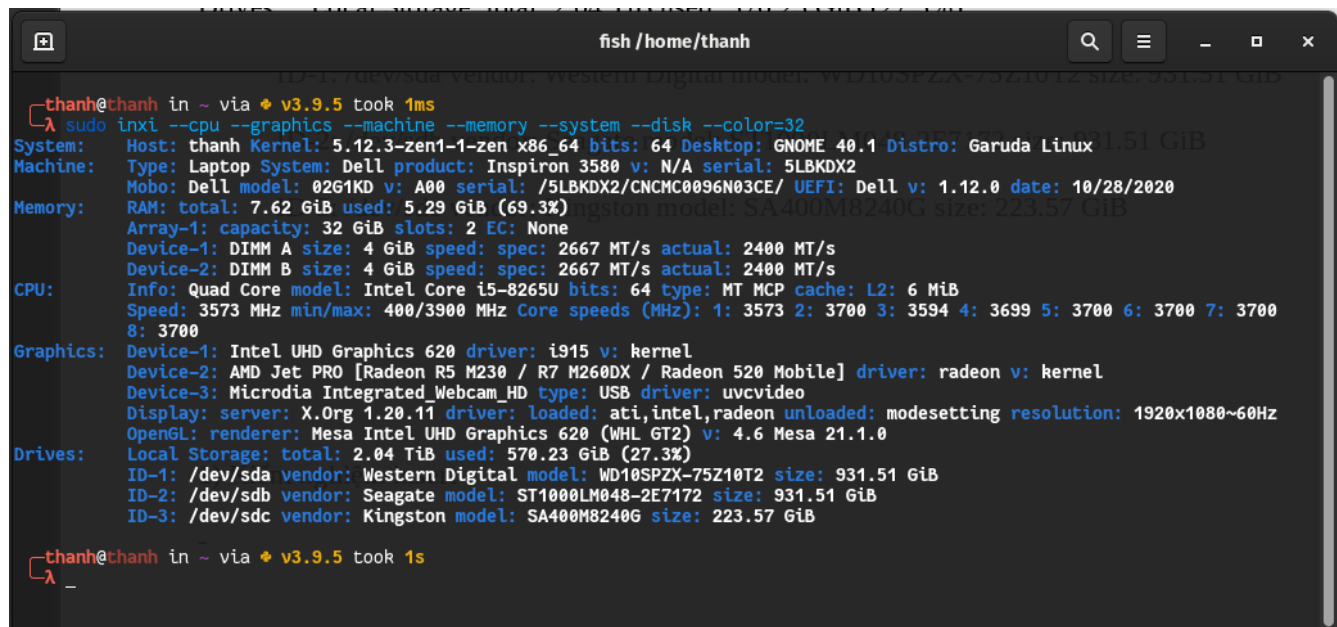
## Drives:

Local Storage: total: 2.04 TiB used: 570.23 GiB (27.3%)

ID-1: /dev/sda vendor: Western Digital model: WD10SPZX-75Z10T2 size: 931.51 GiB

ID-2: /dev/sdb vendor: Seagate model: ST1000LM048-2E7172 size: 931.51 GiB

ID-3: /dev/sdc vendor: Kingston model: SA400M8240G size: 223.57 GiB



```
thanh@thanh in ~ via v3.9.5 took 1ms
λ sudo inxi --cpu --graphics --machine --memory --system --disk --color=32
System: Host: thanh Kernel: 5.12.3-zen1-1-zen x86_64 bits: 64 Desktop: GNOME 40.1 Distro: Garuda Linux 1.51 GiB
Machine: Type: Laptop System: Dell product: Inspiron 3580 v: N/A serial: 5LBKDX2
Mobo: Dell model: 0261KD v: A00 serial: /5LBKDX2/CNCMC0096N03CE/ UEFI: Dell v: 1.12.0 date: 10/28/2020
Memory: RAM: total: 7.62 GiB used: 5.29 GiB (69.3%)
Array-1: capacity: 32 GiB slots: 2 EC: None
Device-1: DIMM A size: 4 GiB speed: spec: 2667 MT/s actual: 2400 MT/s
Device-2: DIMM B size: 4 GiB speed: spec: 2667 MT/s actual: 2400 MT/s
CPU: Info: Quad Core model: Intel Core i5-8265U bits: 64 type: MT MCP cache: L2: 6 MiB
Speed: 3573 MHz min/max: 400/3900 MHz Core speeds (MHz): 1: 3573 2: 3700 3: 3594 4: 3699 5: 3700 6: 3700 7: 3700
8: 3700
Graphics: Device-1: Intel UHD Graphics 620 driver: i915 v: kernel
Device-2: AMD Jet PRO [Radeon R5 M230 / R7 M260DX / Radeon 520 Mobile] driver: radeon v: kernel
Device-3: Microdia Integrated_Webcam_HD type: USB driver: uvcvideo
Display: server: X.Org 1.20.11 driver: loaded: ati,intel,radeon unloaded: modesetting resolution: 1920x1080~60Hz
OpenGL: renderer: Mesa Intel UHD Graphics 620 (WHL GT2) v: 4.6 Mesa 21.1.0
Drives: Local Storage: total: 2.04 TiB used: 570.23 GiB (27.3%)
ID-1: /dev/sda vendor: Western Digital model: WD10SPZX-75Z10T2 size: 931.51 GiB
ID-2: /dev/sdb vendor: Seagate model: ST1000LM048-2E7172 size: 931.51 GiB
ID-3: /dev/sdc vendor: Kingston model: SA400M8240G size: 223.57 GiB
thanh@thanh in ~ via v3.9.5 took 1s
λ _
```

## 2) Cấu hình ổ cứng được sử dụng cho quá trình kiểm nghiệm

Dung lượng lưu trữ: 1TB

- Thương hiệu: Seagate Barracuda
- Tốc độ vòng quay: 5400 RPM
- Bộ nhớ cache: 128MB
- Chuẩn kết nối: SATA III 6Gbps
- Kích thước: 2.5"



### 3) Mô tả workspace

#### 3.1) Quá trình test.

- Trình tự thực hiện hành động:
  - **OPEN => SEARCH => INSERT => UPDATE => BETWEEN => RANK => DELETE**
- Dùng **make** để tạo database mẫu: cú pháp **make <nhãn> [SIZE=<một con số>]**
  - **<nhãn>**: gồm **all**, **SQL**, **database**, **auto**.
    - **all**: chạy 2 nhãn SQL, database.
    - **SQL**: build và run file create\_sql\_files.cpp, xóa file thực thi của file create\_sql\_files.cpp sau khi build.
    - **database**: để chạy nhãn này cần truyền SIZE=<một con số>, chạy file CreateDatabase với đối số SIZE.
    - **auto**: thực hiện toàn bộ quá trình kiểm nghiệm.
  - **[SIZE=<một con số>]**: chỉ truyền khi chạy nhãn **database**, các con số sau sẽ được truyền vào trong quá trình thực hiện kiểm nghiệm.
    - SIZE=250000000, SIZE=500000000, SIZE=750000000, SIZE=1000000000, SIZE=1250000000, SIZE=1500000000, SIZE=1750000000, SIZE=2000000000.

- Thực hiện chạy với 8 database có kích thước tăng dần: 250 triệu, 500 triệu, 750 triệu, 1 tỷ, 1.25 tỷ, 1.5 tỷ, 1.75 tỷ, 2 tỷ, các database này được lưu vào các file **database\_"SIZE".db**, với SIZE là kích thước tương ứng của database, các database này chỉ chứa duy nhất 1 table tên **Btree**, table này chỉ chứa duy nhất 1 cột có kiểu INTEGER. mỗi record chứa 1 số, và giá trị thuộc đoạn **[1, SIZE]**.
- Sau khi tạo database mới thì **phải xóa database cũ đi** để đỡ tốn dung lượng đĩa và để đo dung lượng ổ đĩa khi thực hiện các hành động trên database mới.
- **Chỉ tạo database mới** khi đã thực hiện xong toàn bộ các hành động **OPEN => SEARCH => INSERT => UPDATE => BETWEEN => RANK => DELETE**.
- **INSERT, UPDATE, DELETE** thực hiện với 50000000 record.
- **BETWEEN, RANK** thực hiện đo thời gian của cùng 1 hành động 100 lần.
- **OPEN, SEARCH** thực hiện đo thời gian của cùng 1 hành động 1000 lần.

### 3.2) Các lệnh để thực hiện kiểm nghiệm khi test từng database.

1. **make SQL.**
2. **make database SIZE=250000000**
3. **bash Btree all 250000000**
4. Xóa database cũ.
5. Quay lại và thực hiện bước 2 với **SIZE** tăng dần, **SIZE** và đối số truyền vào sau **all** của lệnh **3** phải giống nhau.
6. Thực hiện 5 bước trên cho đến khi đủ 8 database.

### 3.3) Lệnh rút gọn.

- Nhóm em sử dụng một nhãn **auto**, nên có thể gõ **make auto** để thực hiện toàn bộ quá trình kiểm nghiệm.

### 3.4) Cấu trúc chức năng các thành phần trong cây thư mục.

- Thư mục **operating\_script** chứa các file shell mà file **Btree.sh** gọi tới để thực thi các hành động tương ứng của sqlite.
  - **insert.sh** nhận 1 đối số SIZE (kích thước của database tương ứng), và thực hiện insert 50000000 record vào database đó.
  - **delete.sh** nhận 1 đối số SIZE (kích thước của database tương ứng), và xóa các số > SIZE trong database đó.
  - **rank.sh** nhận 1 đối số SIZE (kích thước của database tương ứng), và thực hiện rank 100 lần trong database đó.
  - **between.sh** nhận 1 đối số SIZE (kích thước của database tương ứng), và thực hiện between 100 lần trong database đó.
  - **update.sh** nhận 1 đối số SIZE (kích thước của database tương ứng), và update các giá trị theo công thức  $n = n + 50000000$  với  $n > SIZE$ .



- **open.sh** nhận 1 đối số SIZE (kích thước của database tương ứng), thực hiện thao tác đóng mở database có kích thước SIZE 1000 lần.
- **search.sh** nhận 1 đối số SIZE (kích thước của database tương ứng), thực hiện tìm kiếm NUMBER = SIZE trong database có kích thước SIZE.
- Thư mục **sheet** chứa các bảng tính thống kê kết quả của các hành động **OPEN, SEARCH, INSERT, UPDATE, BETWEEN, RANK, DELETE**.
- Thư mục **sql** chứa các file .sql được chứa trong các thư mục có tên tương ứng với các hành động.
  - Các thư mục có tên là các con số dùng để chứa các file .sql dùng để thao tác với database có SIZE tương ứng.
  - Ví dụ thư mục 250000000 dùng để chứa các file .sql dùng để thao tác với database có 250000000 record.
- Thư mục **result** dùng để lưu kết quả vào các thư mục tương ứng sau khi chạy các hành động **OPEN, SEARCH, INSERT, UPDATE, BETWEEN, RANK, DELETE**.
  - Các thư mục tương ứng này được chứa trong các thư mục có tên là các con số tương ứng với SIZE của database được thao tác.
  - Ví dụ thư mục 250000000 dùng để chứa các file kết quả sau khi thao tác với database có 250000000 record.
- **Makefile** dùng để thực hiện toàn bộ quá trình thử nghiệm, build file .cpp và thực thi file CreateDatabase.sh.
- **create\_sql\_files.cpp** dùng để tạo ra các file .sql và lưu vào thư mục **sql**.
- **CreateDatabase.sh** nhận 1 đối số là SIZE (kích thước của database tương ứng), và dùng để tạo ra một database mẫu có số lượng record đúng bằng SIZE.
- **measure\_ram.sh** nhận 2 đối số là SIZE (kích thước của database tương ứng) và OPERATION (hành động thực thi), dùng để đo lượng Ram và dung lượng đĩa cứng được sử dụng bởi sqlite khi thực thi OPERATION trên database có SIZE record.
- **recovery.sh** nhận 1 đối số là SIZE (kích thước của database tương ứng), và dùng để khôi phục database có kích thước ban đầu trước khi thực hiện các thao tác.
- **Btree.sh** nhận 2 đối số là SIZE (kích thước của database tương ứng) và OPERATION (hành động thực thi), đây là file shell chính, nó thực thi OPERATION với database có SIZE record, các đối số có thể truyền **insert, delete, update, rank, between, open, search, all**.
  - vd: **bash Btree.sh insert 250000000** -> thực hiện insert vào database có 250000000 record.

operating_script	Delete count.sh	2 hours ago
result	save results of all operations of database which has 1750000000 records	6 hours ago
sheet	search operation sheet	6 hours ago
sql	Save SQL files of corresponding operations in corresponding directories	3 days ago
Btree.sh	Main shell script file to run all operations	6 hours ago
CreateDatabase.sh	shell script file for creating database	6 hours ago
Makefile	Run everything, build .cpp file, run CreateDatabase.sh	6 hours ago
README.md	update README.md	5 hours ago
create_sql_files.cpp	.cpp file that is used for creating sql files corresponding operations	6 hours ago
measure_ram.sh	shell script for measuring ram and disk usage	6 hours ago
recovery.sh	shell script for recovering database	6 hours ago

☰ README.md

#### 4) Quá trình kiểm nghiệm:

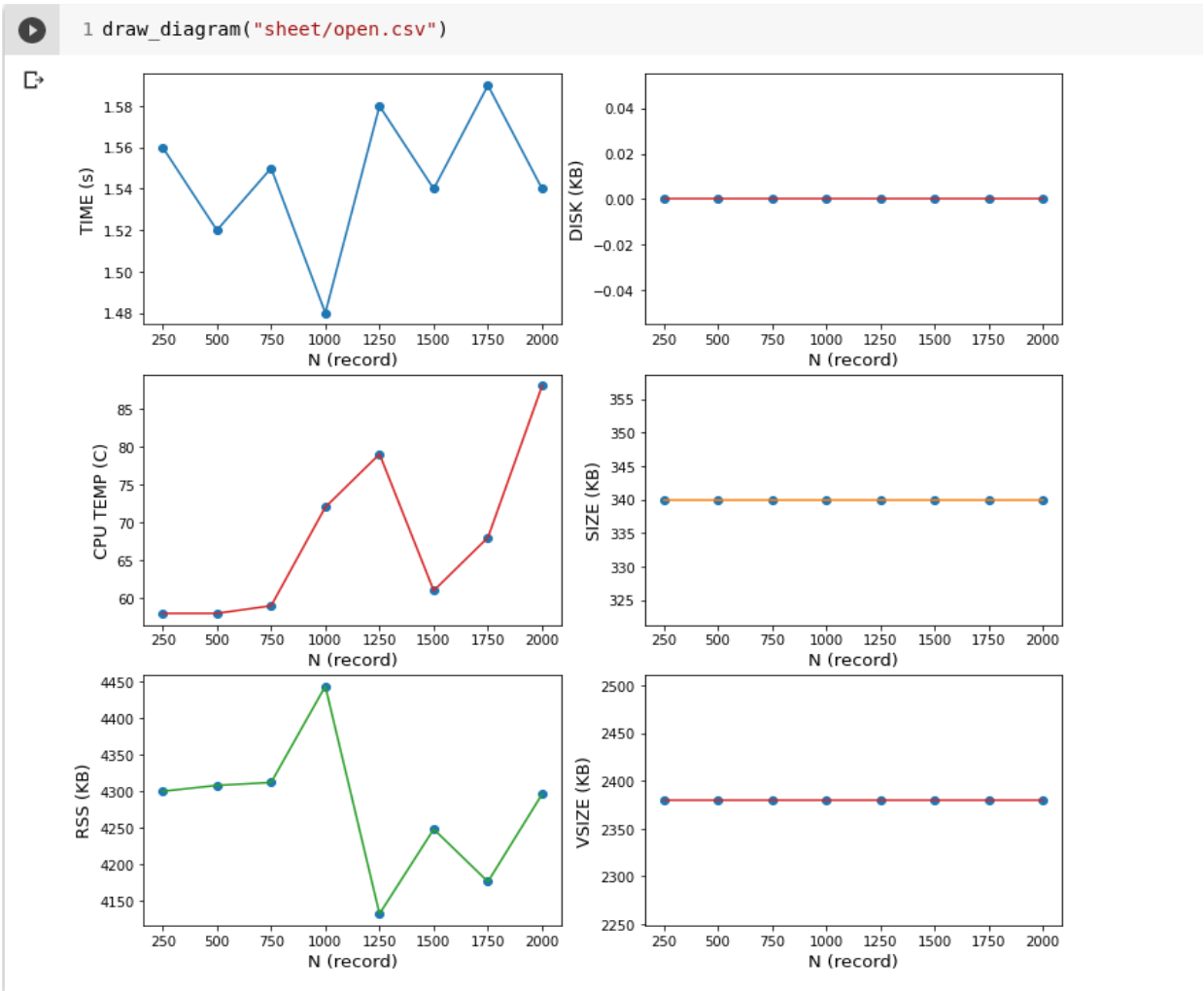
- Để việc kiểm thử được trở nên dễ dàng hơn thì nhóm em sử dụng C++, Shell, Cmake.
- Shell được bọn em sử dụng để đo dung lượng ram và dung lượng đĩa cứng được sử dụng khi kiểm nghiệm, đồng thời tạo database và thực hiện các hành động của sqlite.
- C++ được sử dụng cho việc tạo ra các file sql, các file sql này được sử dụng xuyên suốt trong quá trình thử nghiệm.
- Cmake được bọn em sử dụng để tối giản việc gõ lệnh trên terminal.
- Nhóm em kiểm thử có 8 database có kích thước lần lượt là: 250 triệu, 500 triệu, 750 triệu, 1 tỷ, 1.25 tỷ, 1.5 tỷ, 1.75 tỷ, 2 tỷ.
- Để ước lượng độ phức tạp của các thao tác trên, nhóm em có nhờ sự trợ giúp từ anh Vũ Ngọc Tú, để sử dụng mã nguồn đo độ phức tạp.
- link colab mã nguồn của anh Tú: <https://colab.research.google.com/drive/1w0t-ePiMEa64BuDl8SvdMIGjtMq2Ez?authuser=1#scrollTo=cLropj-GkqdH>

##### 4.1) Open operation

- Thực hiện đóng mở từng database 1000 lần.
- Ước tính độ phức tạp:

Sau khi thực hiện thao tác open với từng database có kích thước tăng dần, nhóm bọn em có được bảng kết quả như sau:

SIZE (triệu record)	Time (s)	Max SIZE (KB)	Max RSS (KB)	VSIZE (KB)	DISK (KB)	CPU Temp (°C)
250	1.56	340	4300	2380	0	58
500	1.52	340	4308	2380	0	58
750	1.55	340	4312	2380	0	59
1000	1.48	340	4444	2380	0	72
1250	1.58	340	4132	2380	0	79
1500	1.54	340	4248	2380	0	61
1750	1.59	340	4176	2380	0	68
2000	1.54	340	4296	2380	0	88



```

O(1)= 0.00105000000000000019
O(lg(n))= 0.021627527325098258
O(sqrt(n))= 0.1792220352982757
O(n)= 0.48294166666666667
O(nlg(n))= 0.5662132107809454
O(n^2)= 0.9590077633378933
O(n^3)= 1.2533852044012492
O(2^n)= 2.388075
O(n*2^n)= 2.388075
O(3^n)= 2.3538033593237273

```

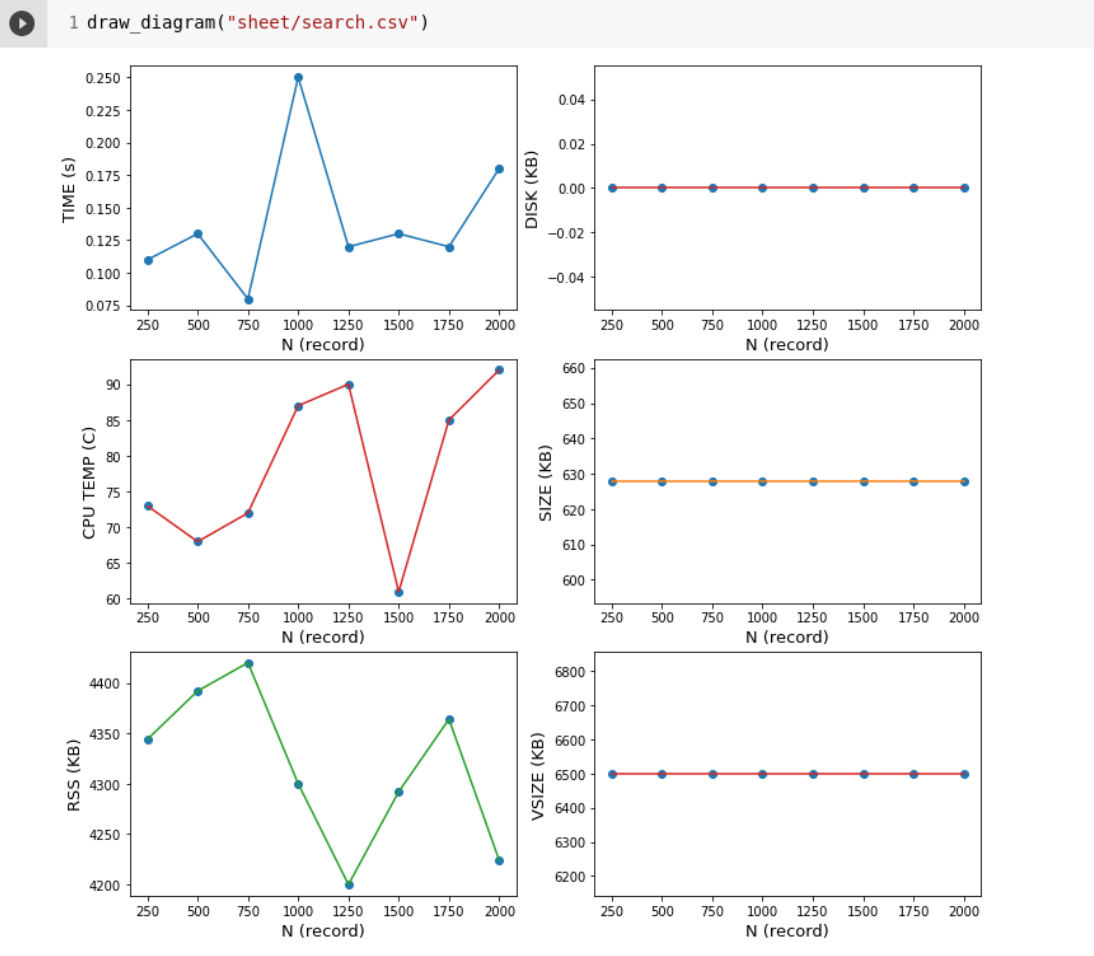
- Thao tác mở database có độ phức tạp gần như tương đương nhau (7/8 database có thời gian mở giao động là 1.5x), đồng thời dung lượng SIZE (memory in Ram) không có biến động.
- Kết luận: Dung lượng Ram cho thao tác open database không phụ thuộc vào kích thước của database, độ phức tạp của thao tác open là  $O(1)$ .

## 4.2. Search operation

- Nhóm em thực hiện thao tác này bằng việc tìm kiếm 1000 lần 1 số cố định có giá trị đúng bằng SIZE của database được thực hiện.
- Ước tính độ phức tạp:

Sau khi thực hiện tìm kiếm với từng database có kích thước tăng dần, nhóm bọn em có được bảng kết quả như sau:

SIZE (triệu record)	Time (s)	Max SIZE (KB)	Max RSS (KB)	VSIZE (KB)	DISK (KB)	CPU Temp (°C)
250	0.11	628	4344	6500	0	73
500	0.13	628	4392	6500	0	68
750	0.08	628	4420	6500	0	72
1000	0.25	628	4300	6500	0	87
1250	0.12	628	4200	6500	0	90
1500	0.13	628	4292	6500	0	61
1750	0.12	628	4364	6500	0	85
2000	0.18	628	4224	6500	0	92



```

O(1)= 0.0024
O(lg(n))= 0.002203758544376567
O(sqrt(n))= 0.0028934477362938006
O(n)= 0.004982291666666667
O(nlg(n))= 0.0056138599669209605
O(n^2)= 0.00880482073643411
O(n^3)= 0.011288174187406592
O(2^n)= 0.022
O(n*2^n)= 0.022
O(3^n)= 0.021991858605936283

```

- Kết luận: Độ phức tạp tìm của thao tác tìm kiếm là  $O(\log(n))$ , dung lượng Ram sử dụng cho quá trình search là một hằng số, và không tăng khi database tăng dần kích thước.

### 4.3. Insert operation

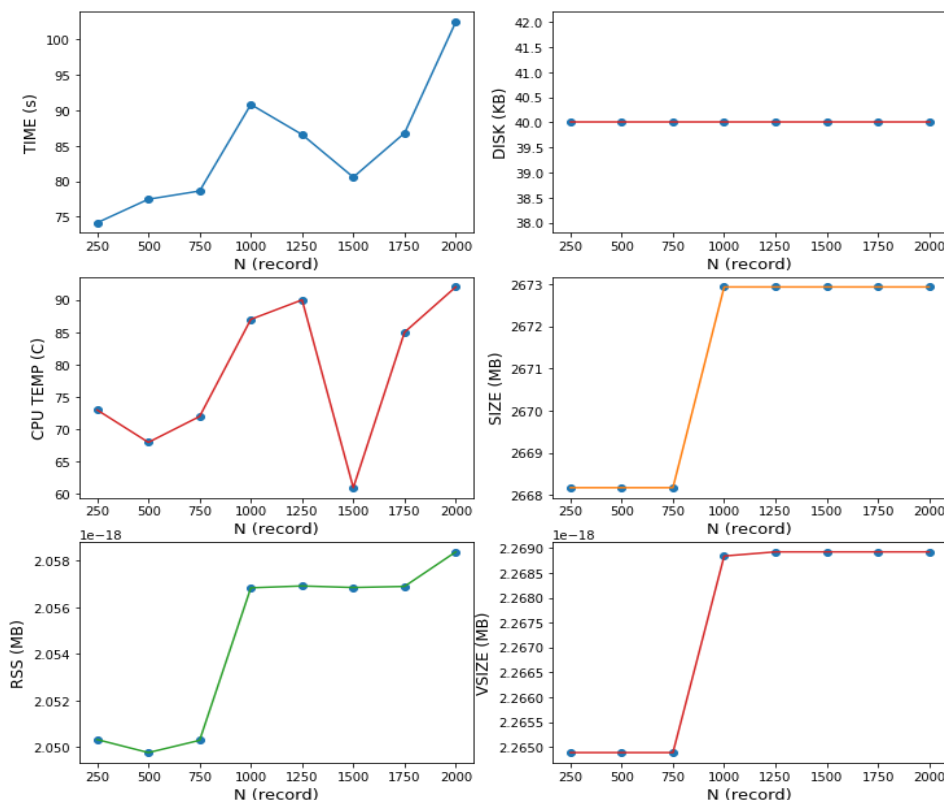
- Thực hiện insert 50000000 số có giá trị khác nhau vào các database, các giá trị được insert thuộc (SIZE, SIZE + 50000000], thao tác này được thực hiện bằng file .sql, 1 lệnh INSERT trong file .sql có 5000000 số nguyên.

- Ước tính độ phức tạp:

Sau khi thực hiện insert với từng database với kích thước tăng dần, nhóm bọn em có được bảng kết quả như sau:

SIZE (triệu record)	Time (s)	Max SIZE(KB)	Max RSS	Max VSIZE	DISK (KB)	CPU temp (°C)
250	74.14	2732208	2478696	2738080	40	73
500	77.46	2732208	2478020	2738080	40	68
750	78.62	2732208	2478656	2738080	40	72
1000	90.84	2737088	2486572	2742860	40	87
1250	86.59	2737088	2486668	2742960	40	90
1500	80.57	2737088	2486588	2742960	40	61
1750	86.8	2737088	2486644	2742960	40	85
2000	102.48	2737088	2488436	2742960	40	92

```
1 draw_diagram("sheet/insert.csv", KB_to_MB=True)
```



```
0(sqrt(n))= 321.4878970515226
0(n)= 1078.9069425857845
0(nlg(n))= 1302.505449175122
0(n^2)= 2408.3583110336867
0(n^3)= 3277.4974232186205
0(2^n)= 7244.443075
0(n*2^n)= 7244.443075
0(3^n)= 7110.126547109079
```

- Kết luận: độ phức tạp của thao tác insert là  $O(\log(n))$ , đồng thời lượng Ram sử dụng cho thao tác này tăng theo kích thước của database.

#### 4.4. Update operation

Thực hiện cập nhật 50000000 số sau khi thực hiện insert, công thức được sử dụng để cập nhật giá trị cho các record là  $n = n + 50000000$ , với  $n > \text{SIZE}$ , SIZE là kích thước ban đầu của database.

- Ước tính độ phức tạp:

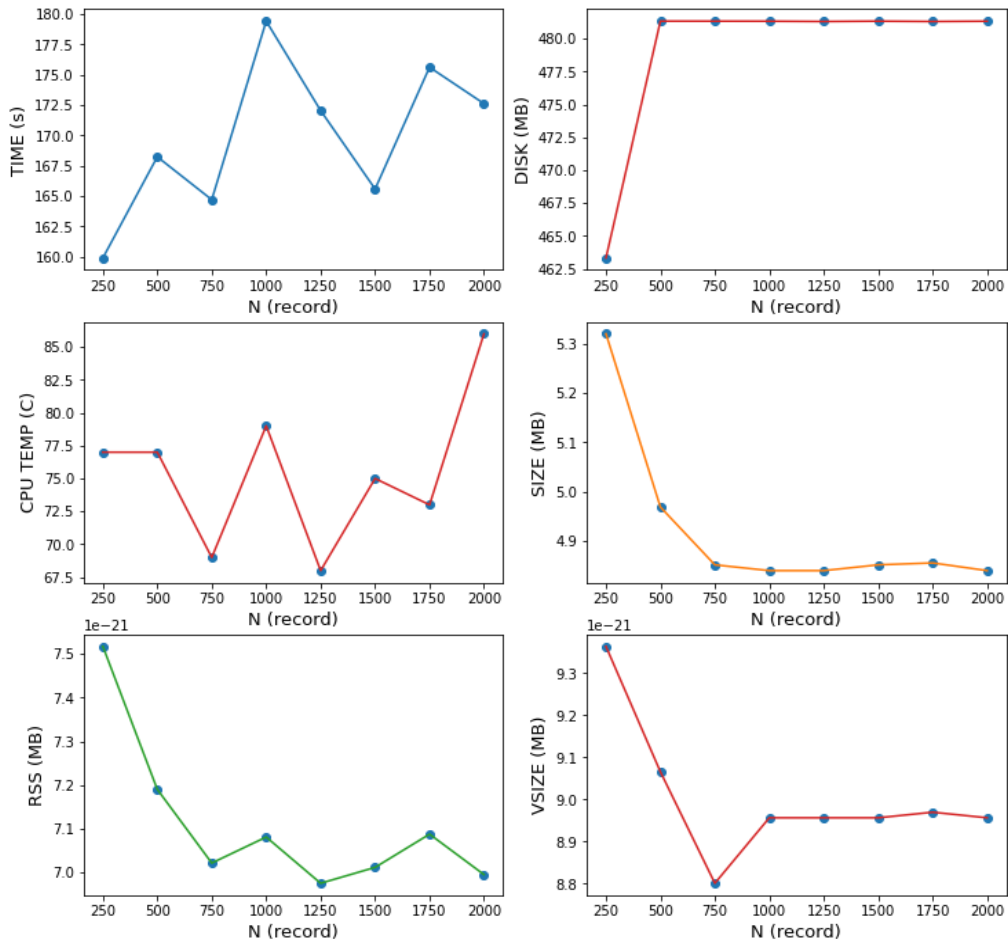
Sau khi thực hiện thao tác update với từng database có kích thước tăng dần, nhóm bọn em có được bảng kết quả như sau:

SIZE (triệu record)	Time (s)	Max SIZE (KB)	Max RSS (KB)	VSIZE (KB)	DISK (KB)	CPU Temp (°C)
250	159.91	5448	9088	11320	474436	77
500	168.24	5088	8692	10960	492900	77
750	164.71	4968	8488	10640	492896	69
1000	179.42	4956	8560	10828	492892	79
1250	172.05	4956	8432	10828	492869	68
1500	165.59	4968	8476	10828	492896	75
1750	175.62	4972	8568	10844	492869	73
2000	172.62	4956	8456	10828	492892	86

```
1 delete = pd.read_csv("sheet/delete.csv").values
2 insert = pd.read_csv("sheet/insert.csv").values
3 update = pd.read_csv("sheet/update.csv").values
4
5 delete = delete[:, 1].reshape(-1, 1)
6 insert = insert[:, 1].reshape(-1, 1)
7 update = update[:, 1].reshape(-1, 1)
8 i = 0
9 while i < 8:
10     Tong = delete[i] + insert[i]
11     Hieu = delete[i] + insert[i] - update[i]
12     print("Tong: ", Tong, " --- ", update[i], " Hieu: ", Hieu)
13     i += 1
```

```
Tong: [171.84] --- [159.91] Hieu: [11.93]
Tong: [195.09] --- [168.24] Hieu: [26.85]
Tong: [207.57] --- [164.71] Hieu: [42.86]
Tong: [228.61] --- [179.42] Hieu: [49.19]
Tong: [217.07] --- [172.05] Hieu: [45.02]
Tong: [220.78] --- [165.59] Hieu: [55.19]
Tong: [228.02] --- [175.62] Hieu: [52.4]
Tong: [258.59] --- [172.62] Hieu: [85.97]
```

```
[14] 1 draw_diagram("sheet/update.csv", KB_to_MB=True)
```



```

O(1)= 72.47041875000004
O(lg(n))= 34.62894359566305
O(sqrt(n))= 321.4878970515226
O(n)= 1078.9069425857845
O(nlg(n))= 1302.505449175122
O(n^2)= 2408.3583110336867
O(n^3)= 3277.4974232186205
O(2^n)= 7244.443075
O(n*2^n)= 7244.443075
O(3^n)= 7110.126547109079

```

- Kết luận: Độ phức tạp của thao tác update là  $O(\log(n))$ , đồng thời dung lượng Ram cũng giảm dần khi thực hiện update (biểu đồ SIZE), tổng thao tác thực hiện insert và delete lớn hơn so với update, nên thao tác update khác với việc delete một record sau đó insert vào.



#### 4.5. Between operation

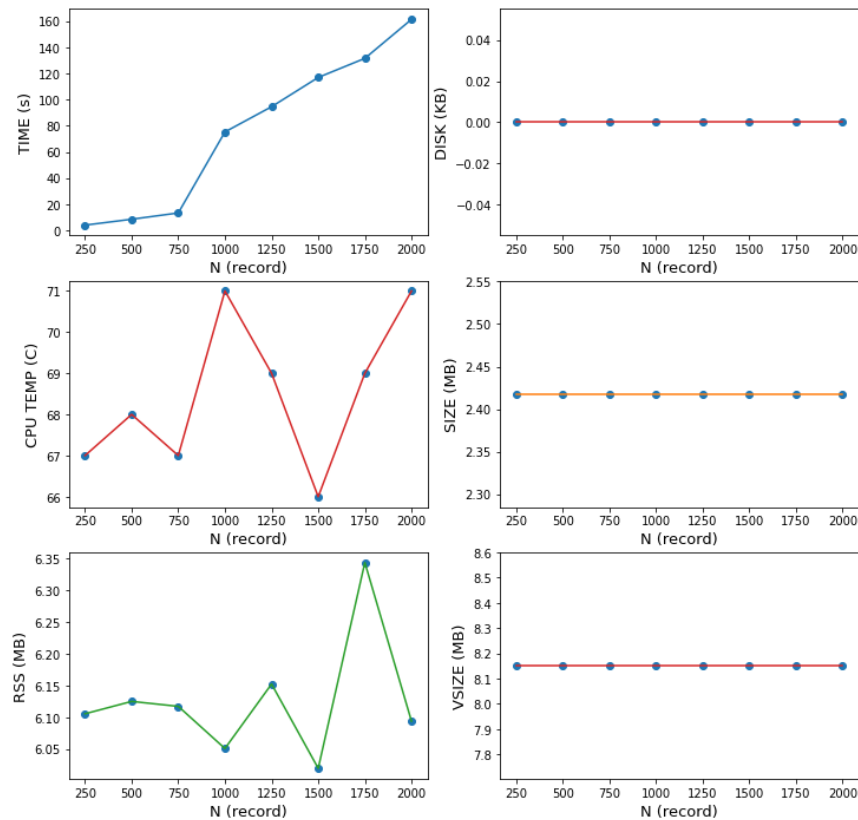
- Thực hiện between 100 lần sau khi thực hiện thao tác update, khoảng giá trị được sử dụng cho các database thuộc  $[1, \text{SIZE} / 2]$  với SIZE là kích thước ban đầu của database đó trước khi thực hiện insert.

- Ước tính độ phức tạp:

Sau khi thực hiện thao tác Between với từng database có kích thước tăng dần, nhóm bọn em có được bảng kết quả như sau:

SIZE (triệu record)	Time (s)	Max SIZE (KB)	Max RSS (KB)	VSIZE (KB)	DISK (KB)	CPU Temp (°C)
250	4.0303	2476	6252	8348	0	67
500	8.5303	2476	6272	8348	0	68
750	13.3427	2476	6264	8348	0	67
1000	75.24	2476	6196	8348	0	71
1250	94.52	2476	6300	8348	0	69
1500	116.88	2476	6164	8348	0	66
1750	131.47	2476	6496	8348	0	69
2000	161.26	2476	6240	8348	0	71

```
[51] 1 draw_diagram("sheet/between.csv", KB_to_MB=True)
```



```

O(1)= 3252.2171147323434
O(lg(n))= 2526.1729978879553
O(sqrt(n))= 1254.5664062657283
O(n)= 376.66019066843126
O(nlg(n))= 265.2370820887495
O(n^2)= 286.0327188127735
O(n^3)= 897.2339420709636
O(2^n)= 8976.525984933749
O(n*2^n)= 8976.525984933749
O(3^n)= 8759.934528368858

```

- Kết luận: độ phức tạp của thao tác Between là  $O(\log(n))$ , đồng thời lượng Ram không sử dụng không thay đổi khi tăng kích thước của database (biểu đồ SIZE), thao tác này không tốn thêm dung lượng ổ đĩa để sinh thêm file tạm.

## 4.6. Rank operation

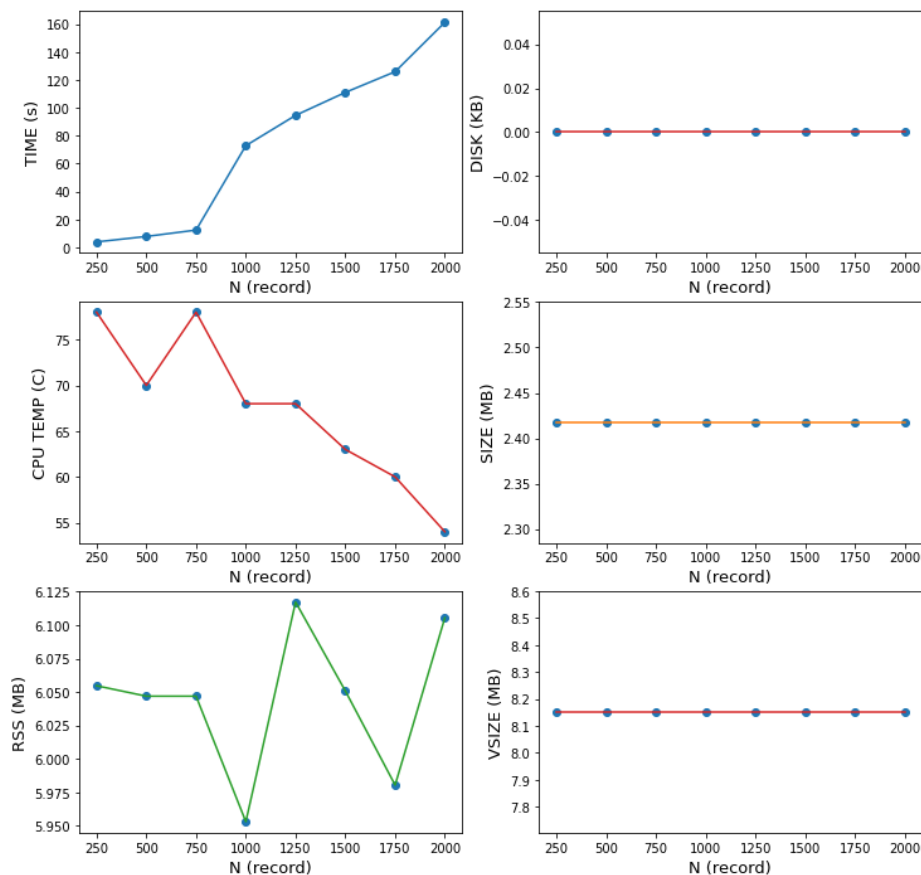
- Thao tác này được thực hiện sau khi Between thực thi xong, nhóm em tiến hành Rank 100 lần với số có giá trị là  $SIZE / 2$ , với SIZE là kích thước ban đầu của database đó trước khi thực hiện insert.

- Ước tính độ phức tạp:

Sau khi thực hiện thao tác Rank với từng database có kích thước tăng dần, nhóm bọn em có được bảng kết quả như sau:

SIZE record)	(triệu Time (s)	Max (KB)	SIZE Max (KB)	RSS VSIZE (KB)	DISK (KB)	CPU (°C)	Temp
250	3.9103	2476	6200	8348	0	78	
500	7.824	2476	6192	8348	0	70	
750	12.3987	2476	6192	8348	0	78	
1000	72.95	2476	6096	8348	0	68	
1250	94.68	2476	6264	8348	0	68	
1500	111.13	2476	6196	8348	0	63	
1750	125.99	2476	6124	8348	0	60	
2000	161.4	2476	6252	8348	0	54	

```
[52] 1 draw_diagram("sheet/rank.csv", KB_to_MB=True)
```



```

O(1)= 3154.412821581875
O(lg(n))= 2459.4196272485733
O(sqrt(n))= 1236.598765896531
O(n)= 382.24163793122557
O(nlg(n))= 271.24724207722505
O(n^2)= 263.88182635831345
O(n^3)= 821.4824352898523
O(2^n)= 8598.6943854725
O(n*2^n)= 8598.6943854725
O(3^n)= 8356.416120924332

```

- Kết luận: độ phức tạp của thao tác Rank là  $O(n^2)$ , đồng thời lượng Ram không sử dụng không thay đổi khi tăng kích thước của database (biểu đồ SIZE), thao tác này không tốn thêm dung lượng ổ đĩa để sinh thêm file tạm.

#### 4.7. Delete operation

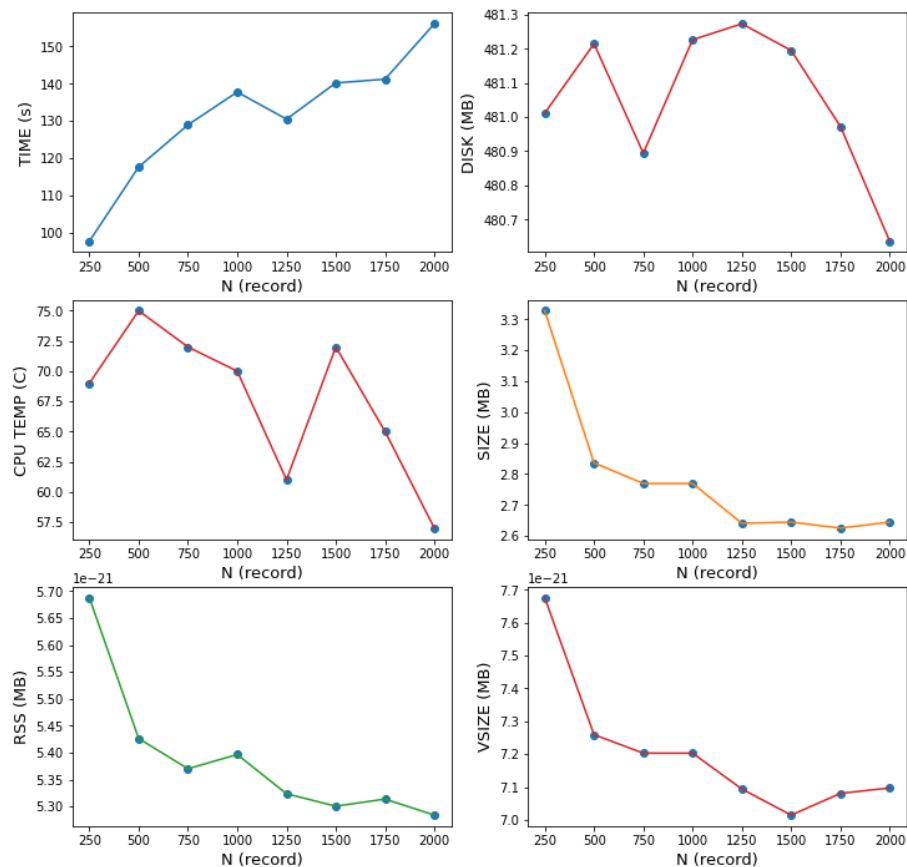
- Nhóm em tiến hành việc xóa 50000000 record sau khi thực hiện thao tác Rank, thao tác này sẽ xóa các record mà có giá trị khóa lớn hơn SIZE, với SIZE là kích thước ban đầu của database đó trước khi thực hiện insert.

- Ước tính độ phức tạp:

Sau khi thực hiện thao tác Delete 50000000 record với từng database có kích thước tăng dần, nhóm bọn em có được bảng kết quả như sau:

SIZE (triệu record)	Time (s)	Max SIZE (KB)	Max RSS (KB)	VSIZE (KB)	DISK (KB)	CPU Temp (°C)
250	97.7	3408	6876	9280	492556	69
500	117.63	2904	6560	8776	492764	75
750	128.95	2836	6492	8708	492436	72
1000	137.77	2836	6524	8708	492776	70
1250	130.48	2704	6436	8576	492824	61
1500	140.21	2708	6408	8480	492744	72
1750	141.22	2688	6424	8560	492516	65
2000	156.11	2708	6388	8580	492172	57

```
[17] 1 draw_diagram("sheet/delete.csv", KB_to_MB=True)
```



```

O(1)= 269.6502109375001
O(lg(n))= 31.407396655695784
O(sqrt(n))= 488.74955307261155
O(n)= 2167.8087550857845
O(nlg(n))= 2690.5746313172444
O(n^2)= 5387.177967320736
O(n^3)= 7582.493370909374
O(2^n)= 17498.5096625
O(n*2^n)= 17498.5096625
O(3^n)= 17226.57235684043

```

- Kết luận: độ phức tạp của thao tác delete là  $O(\log(n))$ , đồng thời dung lượng Ram cũng giảm dần khi thực hiện delete (biểu đồ SIZE) do có nhiều phần tử đã bị xóa đi khỏi database.

- Link github: <https://github.com/Nhat-Thanh/CS523.L21/tree/main/Btree>

- Link colab của nhóm em: [https://colab.research.google.com/drive/15-eo0HgmPYuE73\\_BmRjd\\_5HZxYTiKxf-?usp=sharing](https://colab.research.google.com/drive/15-eo0HgmPYuE73_BmRjd_5HZxYTiKxf-?usp=sharing)