

**BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NHA TRANG
KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO THỰC TẬP CƠ SỞ
MÔ PHỎNG CÁC THUẬT TOÁN SẮP XẾP BẰNG ĐỒ HỌA**

Sinh viên thực hiện: Phạm Tuấn Kiệt

Mã số sinh viên: 64131060

Lớp 64.CNTT-4

Giảng viên hướng dẫn: Nguyễn Mạnh Cường

Nha Trang – Khánh Hòa – Tháng 12/2024

LỜI CẢM ƠN

Trước tiên, em xin gửi lời cảm ơn chân thành và sâu sắc đến các thầy cô trong khoa công nghệ thông tin của trường đại học Nha Trang, đã tận tình truyền đạt các kiến thức liên quan từ cơ bản đến nâng cao cho đến thời điểm hiện tại, để em có thể thực hiện đề tài này.

Đặc biệt, em xin gửi lời cảm ơn đến **Thầy Nguyễn Mạnh Cường**, người đã trực tiếp hướng dẫn và hỗ trợ em không những trong môn lập trình Python, mà còn chia sẻ những kinh nghiệm, những hướng đi cụ thể để em thực hiện đề tài và bài báo cáo này. Những góp ý, định hướng của thầy đã giúp em rất nhiều trong việc tìm hiểu và hoàn thành đề tài này một cách hiệu quả.

Cuối cùng, với những kinh nghiệm về thực tế và kiến thức còn hạn chế nên sản phẩm cuối cùng và bài báo cáo của em có thể chưa hoàn chỉnh và có nhiều sai sót, rất mong thầy cô có thể đóng góp ý kiến cho em để em tiếp thu và sửa đổi cho các đề tài khác.

Em xin chân thành cảm ơn thầy cô.

Nha Trang, ngày 4 tháng 1 năm 2025

Sinh viên thực hiện



Phạm Tuấn Kiệt

MỤC LỤC

DANH MỤC HÌNH ẢNH.....	1
DANH MỤC BẢNG.....	2
Chương I: GIỚI THIỆU CHUNG.....	3
1.1 Lý do chọn đề tài.....	3
1.2 Mục tiêu đề tài.....	3
1.3 Cấu trúc báo cáo.....	3
Chương II: CƠ SỞ LÝ THUYẾT.....	5
2.1 Các thuật toán sắp xếp.....	5
2.1.1 Sắp xếp chèn – Insertion Sort.....	5
2.1.2 Sắp xếp chọn – Selection Sort.....	6
2.1.3 Sắp xếp nổi bọt – Bubble Sort.....	8
2.1.4 Sắp xếp trộn – Merge Sort.....	9
2.1.5 Sắp xếp nhanh – Quick Sort.....	10
2.1.6 Sắp xếp vun đống – Heap Sort.....	11
2.1.7 Đặc điểm của các thuật toán sắp xếp	14
2.2 Ngôn ngữ Python	15
2.2.1 Giới thiệu chung về Python.....	15
2.2.2 Đặc điểm của ngôn ngữ Python	16
2.3 Thư viện Tinker	16
2.3.1 Giới thiệu chung.....	16
2.3.2 Các Widget trong Tkinter của Python.....	17
2.3.3 Tạo giao diện cho người dùng tương tác	20
2.3.4 Tạo các cột số.....	20
2.4 Các thư viện hỗ trợ khác	21
2.4.1 Thư viện Threading	21
2.4.2 Thư viện Time	22
2.4.3 Thư viện Random.....	22
Chương III: PHÂN TÍCH THIẾT KẾ CHƯƠNG TRÌNH	23
3.1 Mô tả hệ thống.....	23

3.2 Các yêu cầu cơ bản của hệ thống	23
Chương IV: THIẾT KẾ CHƯƠNG TRÌNH.....	24
4.1 Tổng quan.....	24
4.2 Module Sort.....	24
4.2.1 Hàm trực quan của sắp xếp chọn	24
4.2.2 Hàm trực quan của sắp xếp chèn.....	24
4.2.3 Hàm trực quan của sắp xếp nổi bọt.....	25
4.2.4 Hàm trực quan của sắp xếp trộn.....	25
4.2.5 Hàm trực quan của sắp xếp vun đống	26
4.2.6 Hàm trực quan của sắp xếp nhanh	27
4.3 Module Main	28
4.3.1 Hàm hiển thị các cột số	28
4.3.2 Hàm tạo mảng số.....	29
4.3.3 Hàm lựa chọn thuật toán	30
4.3.4 Hàm cho phép tạm dừng chương trình.....	31
4.3.5 Giao diện tương tác với người dùng	31
Chương V: KẾT LUẬN	33
5.1 Ưu điểm – Kết quả đạt được	33
5.2 Nhược điểm – Hạn chế.....	33
5.3 Kết luận	33
TÀI LIỆU THAM KHẢO	34

DANH MỤC HÌNH ẢNH

Hình 2.1: Sắp xếp chèn trong thực tế.....	5
Hình 2.2: Cha đẻ của ngôn ngữ Python	15
Hình 2.3: Thư viện Tkinter.....	17
Hình 2.4: Minh họa Label trong Tkinter.....	17
Hình 2.5: Minh họa Button trong Tkinter	18
Hình 2.7: Minh họa OptionMenu trong Tkinter	19
Hình 2.8: Một số Widget khác của Tkinter.....	19
Hình 2.9: Giao diện để tương tác với người dùng	20
Hình 2.10: Mô phỏng các cột số	21
Hình 2.11: Mô phỏng chạy đa luồng.....	21
Hình 3.1: Sơ đồ phân rã chức năng.....	23
Hình 4.1: Hàm selection_sort.....	24
Hình 4.2: Hàm insertion_sort.....	25
Hình 4.3: Hàm bubble_sort.....	25
Hình 4.4: Hàm merge_sort	25
Hình 4.5: Hàm merge_sort_alg	26
Hình 4.6: Hàm merge	26
Hình 4.7: Hàm heapfy	27
Hình 4.8: Hàm heap_sort	27
Hình 4.9: Hàm partition	28
Hình 4.10: Hàm quick_sort.....	28
Hình 4.11: Hàm Display.....	29
Hình 4.12: HàmCreate_Array tạo mảng số ngẫu nhiên.....	29
Hình 4.13: Hàm Input_Array cho phép nhập mảng mong muốn	30
Hình 4.14: Hàm Sort_Algorithm	30
Hình 4.15: Hàm toggle_pause.....	31
Hình 4.16: Giao diện tương tác chính của chương trình.....	32

DANH MỤC BẢNG

Bảng 2.1: Minh họa thuật toán sắp xếp chèn	6
Bảng 2.2: Minh họa thuật toán sắp xếp chọn.....	8
Bảng 2.3: Minh họa thuật toán sắp xếp nổi bọt	9
Bảng 2.4: Minh họa thuật toán sắp xếp trộn	10
Bảng 2.5: Minh họa thuật toán sắp xếp nhanh.....	11
Bảng 2.6: Minh họa thuật toán sắp xếp vun đống.....	14
Bảng 2.7: Đặc điểm của 6 thuật toán sắp xếp	15
Bảng 2.8: Các phiên bản Python.....	15

Chương I: GIỚI THIỆU CHUNG

1.1 Lý do chọn đề tài

Các thuật toán sắp xếp là một trong những kiến thức cơ bản nhưng lại đóng vai trò quan trọng đối với người học công nghệ thông tin, đặc biệt là sinh viên. Hiểu rõ các thuật toán sắp xếp giúp sinh viên rèn luyện tư duy tính toán, kỹ năng phân tích và so sánh hiệu suất của các phương pháp khác nhau, từ đó hỗ trợ sinh viên trong nhận dạng được các vấn đề và đưa ra cách giải quyết vấn đề một cách hiệu quả nhất trong thời gian ngắn nhất.

Việc mô phỏng các thuật toán sắp xếp bằng đồ họa đóng vai trò quan trọng trong việc nâng cao chất lượng giảng dạy và học tập. Thay vì sử dụng những hình ảnh thô sơ, cứng nhắc như trước đây, mô phỏng đồ họa sẽ tạo ra một môi trường học tập sinh động, trực quan, giúp người học dễ dàng hình dung cách các thuật toán hoạt động từng bước. Điều này không chỉ hỗ trợ giảng viên truyền đạt kiến thức một cách nhanh chóng và hiệu quả hơn, mà còn giúp giảm bớt sự nhàm chán, tạo hứng thú cho sinh viên khi học tập.

Bên cạnh đó, đối với sinh viên, mô phỏng bằng đồ họa giúp họ tiếp thu cơ chế hoạt động của các thuật toán một cách dễ dàng và rõ ràng hơn. Những biểu diễn trực quan này không chỉ giúp họ hiểu nhanh hơn mà còn khuyến khích tư duy phản biện và khả năng so sánh, đánh giá hiệu quả của từng thuật toán trong các tình huống khác nhau. Quan trọng hơn, công cụ này còn đáp ứng nhu cầu tự học của sinh viên, giúp họ chủ động tìm hiểu và khám phá các thuật toán sắp xếp một cách độc lập, từ đó xây dựng nền tảng vững chắc cho những kiến thức chuyên sâu hơn.

1.2 Mục tiêu đề tài

- Tạo ra một chương trình mô phỏng 6 thuật toán sắp xếp bao gồm:
 - + Sắp xếp chèn (Insertion Sort)
 - + Sắp xếp chọn (Selection Sort)
 - + Sắp xếp nổi bọt (Bubble Sort)
 - + Sắp xếp trộn (Merge sort)
 - + Sắp xếp nhanh (Quick sort)
 - + Sắp xếp vun đống (Heap sort)
- Chương trình mô phỏng 6 thuật toán bằng hình ảnh, chuyển động, màu sắc dễ nhìn, dễ hiểu.
- Chương trình sử dụng Python có thể sử dụng Tkinter biểu diễn các số và thiết kế giao diện người dùng. Bên cạnh đó, còn có các thư viện hỗ trợ như Random, Time, Threading.

1.3 Cấu trúc báo cáo

Chương I: Giới thiệu chung

Chương II: Cơ sở lý thuyết

Chương III: Phân tích thiết kế chương trình

Chương IV: Thiết kế chương trình

Chương V: Kết luận

Chương II: CƠ SỞ LÝ THUYẾT

2.1 Các thuật toán sắp xếp

2.1.1 Sắp xếp chèn – Insertion Sort

Sắp xếp chèn (Insertion Sort) là thuật toán sắp xếp đơn giản hoạt động bằng cách lặp lại việc chèn từng phần tử của một danh sách chưa được sắp xếp vào vị trí chính xác của nó trong một phần đã được sắp xếp của danh sách. Nó giống như việc sắp xếp các lá bài trong tay bạn. Bạn chia các nhóm lá bài thành hai: các lá bài đã được sắp xếp và các lá bài chưa được sắp xếp. Sau đó, bạn chọn một lá bài từ nhóm chưa được sắp xếp và đặt nó vào vị trí chính xác trong nhóm đã được sắp xếp [7].



Hình 2.1: Sắp xếp chèn trong thực tế

Thuật toán sắp xếp chèn, ta sẽ xem xét từ đầu danh sách tới cuối, bắt đầu từ phần tử thứ hai trong danh sách, vì phần tử đầu tiên coi như đã được sắp xếp. Đầu tiên, thuật toán lấy phần tử hiện tại (giả sử là phần tử thứ i), và so sánh nó với phần tử ngay trước đó (phần tử thứ $i-1$). Nếu phần tử hiện tại nhỏ hơn phần tử phía trước, thuật toán sẽ dịch chuyển phần tử phía trước sang phải để tạo không gian cho phần tử hiện tại. Quá trình so sánh và dịch chuyển này tiếp tục cho đến khi tìm được vị trí thích hợp để chèn phần tử vào. Sau đó, thuật toán lặp lại quá trình này cho đến khi toàn bộ danh sách được duyệt qua và sắp xếp hoàn chỉnh.

Ví dụ với một mảng có các phần tử 8, 5, 2, 7, 9, 3 [1]

Phần tử thứ nhất coi như là đã sắp xếp.	8	5	2	7	9	3
---	---	---	---	---	---	---

Phần tử thứ 2, số 5 bé hơn số 8, đổi chỗ số 5 ra trước số 8.	
Phần tử thứ 3, số 2 bé hơn số 8, đổi chỗ số 2 ra trước số 8. Số 2 vẫn bé hơn số 5, đổi chỗ cho số 2 ra trước số 5.	
Phần tử thứ 4, số 7 bé hơn số 8, đổi chỗ số 7 ra trước số 8. Số 7 lớn hơn số 2 và số 5 nên phần tử này đứng yên.	
Phần tử thứ 5, số 9 lớn hơn các phần tử trước nên phần tử này đứng yên.	
Phần tử thứ 6, số 3 bé hơn các số 9; 8; 7 và số 5, nên số 3 sẽ đổi chỗ lần lượt qua trước số 9, số 8 cứ tiếp tục cho đến khi trước đứng số.	
Kết quả sắp xếp chèn.	

Bảng 2.1: Minh họa thuật toán sắp xếp chèn

2.1.2 Sắp xếp chọn – Selection Sort

Sắp xếp chọn (Selection Sort) là thuật toán sắp xếp bằng cách chọn phần tử nhỏ nhất hoặc lớn nhất từ mảng ban đầu chưa được sắp xếp và hoán đổi nó với phần tử chưa được sắp xếp đầu tiên. Quá trình này tiếp tục cho đến khi toàn bộ mảng được sắp xếp [7].

Đầu tiên chúng ta tìm phần tử nhỏ nhất và hoán đổi nó với phần tử đầu tiên. Bằng cách này, chúng ta sẽ có được phần tử nhỏ nhất ở đúng vị trí của nó. Sau đó, chúng

ta tìm phần tử nhỏ nhất trong số các phần tử còn lại (hoặc phần tử nhỏ thứ hai) và di chuyển nó đến vị trí chính xác bằng cách hoán đổi. Chúng ta tiếp tục làm như vậy cho đến khi di chuyển được tất cả các thành phần đến đúng vị trí.

Ví dụ với một mảng có các phần tử 8, 5, 2, 7, 9, 3 [1]

Đầu tiên, ta duyệt qua các phần tử để tìm phần tử nhỏ nhất của mảng	
Ta thấy phần tử nhỏ nhất có giá trị là 2, ta đổi chỗ 2 với phần tử đầu tiên là 8.	
Ta tiếp tục tìm phần tử có giá trị nhỏ nhất từ phần tử thứ 2 trở đi, ta thấy phần tử nhỏ nhất có giá trị nhỏ nhất là 3, ta đổi chỗ cho phần tử đầu tiên của phần chưa sắp xếp là 5.	
Tiếp tục, ta tìm được phần tử có giá trị nhỏ nhất là 5, ta đổi chỗ với phần tử đầu tiên của phần chưa sắp xếp là 8.	
Tiếp theo, phần tử có giá trị nhỏ nhất là 7, vì 7 đã được sắp xếp theo đúng thứ tự nên sẽ bỏ qua phần tử này.	

Tiếp theo, phần tử có giá trị nhỏ nhất là 8, ta đổi chỗ với phần tử đầu tiên của phần chưa sắp xếp là 9.	
Kết quả sắp xếp chọn	

Bảng 2.2: Minh họa thuật toán sắp xếp chọn

2.1.3 Sắp xếp nổi bọt – Bubble Sort

Sắp xếp nổi bọt (Bubble Sort) là thuật toán sắp xếp đơn giản nhất hoạt động bằng cách hoán đổi liên tục các phần tử liền kề nếu chúng không đúng thứ tự. Thuật toán này không phù hợp với các tập dữ liệu lớn vì độ phức tạp thời gian trung bình và trường hợp xấu nhất của nó khá cao [7].

Thuật toán này xuất phát từ phần tử cuối (đầu tiên) danh sách ta tiến hành so sánh với phần tử bên trái (phải) của nó. Nếu phần tử đang xét có khóa nhỏ hơn phần tử bên trái (phải) của nó ta tiến đưa nó về bên trái (phải) của dãy bằng cách hoán vị với phần tử bên trái (phải) của nó. Tiếp tục thực hiện như thế đối với bài toán có n phần tử thì sau $n - 1$ bước ta thu được danh sách tăng dần [3].

Ví dụ với một mảng có các phần tử 8, 5, 2, 7, 9, 3 [1]

Lần 1	
-------	--

Lần 2	
Lần 3	
Lần 4	
Lần 5	

Bảng 2.3: Minh họa thuật toán sắp xếp nổi bọt

2.1.4 Sắp xếp trộn – Merge Sort

Sắp xếp trộn (Merge Sort) là thuật toán sắp xếp theo phương pháp chia để trị. Thuật toán này hoạt động bằng cách đệ quy chia mảng đầu vào thành các mảng con nhỏ hơn và sắp xếp các mảng con đó sau đó hợp nhất chúng lại với nhau để thu được mảng đã sắp xếp [7].

Đối với thuật toán Merge Sort - sắp xếp trộn, chúng ta sẽ thực hiện chia đôi liên tục dãy lớn thành các dãy con, cho đến khi ta thu được các dãy chỉ bao gồm một phần tử. Sau đó sắp xếp lại các phần tử bắt đầu từ những dãy con nhỏ nhất. Cuối cùng thực hiện thao tác gộp lần lượt ngược lại các dãy con để trở về dãy với số phần tử như ban đầu đã được sắp xếp [4].

Ví dụ với một mảng A có các phần tử 8, 5, 2, 7, 9, 3 [1]

Ta chia mảng A ban đầu thành 2 mảng con nhỏ hơn	
---	--

Ta tiếp tục chia nhỏ các mảng con thành các mảng nhỏ hơn	
Ta tiếp tục chia nhỏ cho đến khi nào không còn chia được nữa	
Sau cùng, ta gộp các mảng con lại để thành dãy với các phần tử ban đầu nhưng đã được sắp xếp	

Bảng 2.4: Minh họa thuật toán sắp xếp trộn

2.1.5 Sắp xếp nhanh – Quick Sort

Sắp xếp nhanh (Quick Sort) là một thuật toán sắp xếp dựa trên nguyên lý chia để trị, chọn một phần tử làm trục và phân mảng đúng cho xung quanh trục đã chọn bằng cách đặt trục vào vị trí của nó trong sắp xếp mảng [7].

Thuật toán này hoạt động bằng cách chọn một phần tử làm chốt (pivot) và sắp xếp các phần tử còn lại dựa trên giá trị của phần tử này. Đầu tiên, mảng được chia thành hai phần: các phần tử nhỏ hơn hoặc bằng pivot nằm bên trái và các phần tử lớn hơn pivot nằm bên phải. Sau đó, thuật toán được áp dụng đệ quy để sắp xếp hai phần mảng này. Quá trình này tiếp tục cho đến khi mỗi phần mảng chỉ còn một phần tử hoặc không còn phần tử nào, lúc này mảng được coi là đã sắp xếp.

Ví dụ với một mảng A có các phần tử 8, 7, 2, 1, 5, 3, 6, 4 [1]

Chọn phần tử chính giữa mảng để làm chốt, lần lượt đổi chỗ các phần tử nhỏ hơn chốt sang bên trái và ngược lại. Lúc này, ta được phần bên trái gồm các phần tử nhỏ hơn chốt và các phần bên phải gồm các phần tử lớn hơn chốt.	
--	--

Vì phần mảng bên trái không có phần tử, ta đệ quy mảng bên phải. Sau khi thực hiện đệ quy mảng bên phải, ta được 2 mảng con trái và phải nhỏ hơn.	
Ta tiếp tục đệ quy 2 mảng con này.	
Sau cùng ta nhận được một dãy đã sắp xếp	

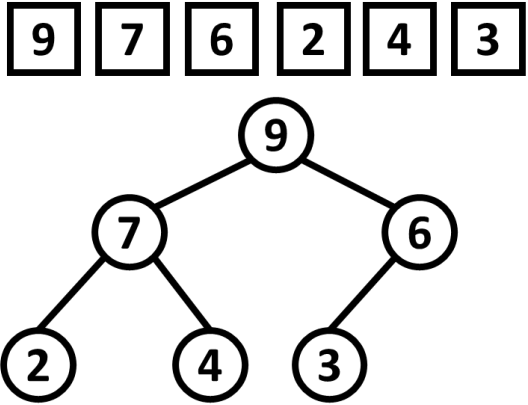
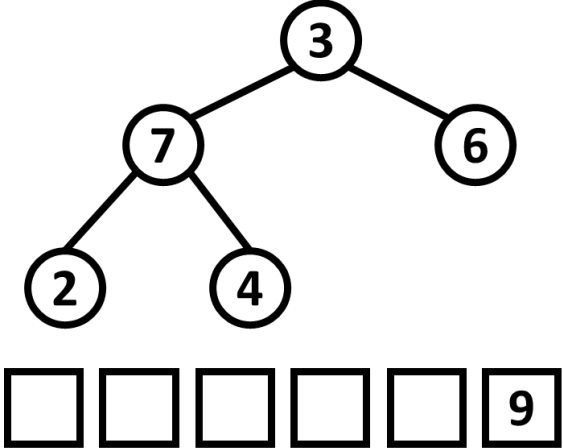
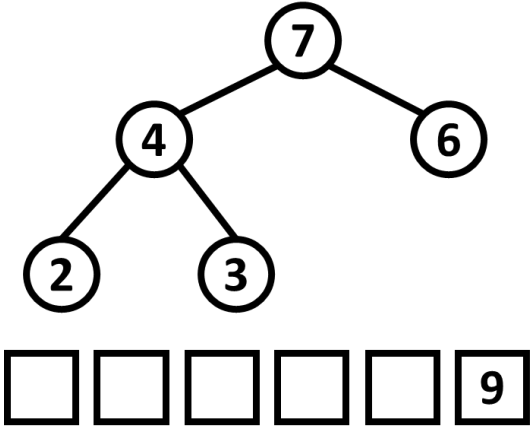
Bảng 2.5: Minh họa thuật toán sắp xếp nhanh

2.1.6 Sắp xếp vun đống – Heap Sort

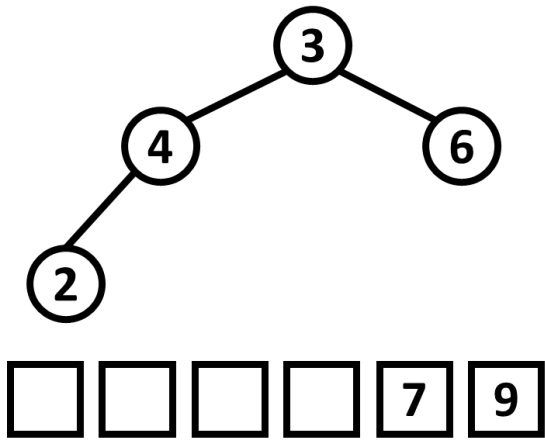
Heap là một cấu trúc dữ liệu dạng cây, trong đó các nút trên cây được sắp xếp theo một thứ tự ràng buộc nhất định giữa khóa của nút cha và khóa của nút con (thường là nút cha nhỏ hơn hoặc lớn hơn nút con). Nút ở gốc của Heap luôn luôn là nút có mức ưu tiên cao nhất, nghĩa là lớn nhất hoặc nhỏ nhất [5].

Sắp xếp (Heap Sort) là một thuật toán sắp xếp sử dụng cấu trúc dữ liệu Heap để tổ chức các phần tử trong mảng. Quá trình hoạt động của thuật toán bắt đầu bằng việc xây dựng một Heap từ mảng đầu vào. Nếu sử dụng Max-Heap, phần tử gốc sẽ là phần tử lớn nhất, trong khi Min-Heap sẽ đưa phần tử nhỏ nhất lên đầu. Sau đó, thuật toán lấy phần tử gốc ra khỏi Heap, đồng thời đẩy phần tử cuối cùng của Heap lên vị trí gốc. Tiếp theo, Heap được sắp xếp lại để đảm bảo tính chất của Heap được duy trì. Quá trình này lặp lại, lần lượt lấy các phần tử gốc ra khỏi Heap và chèn chúng vào cuối dãy đã sắp xếp. Khi tất cả các phần tử đã được lấy ra khỏi Heap, mảng đầu vào sẽ được biến đổi thành một dãy đã sắp xếp hoàn chỉnh [6].

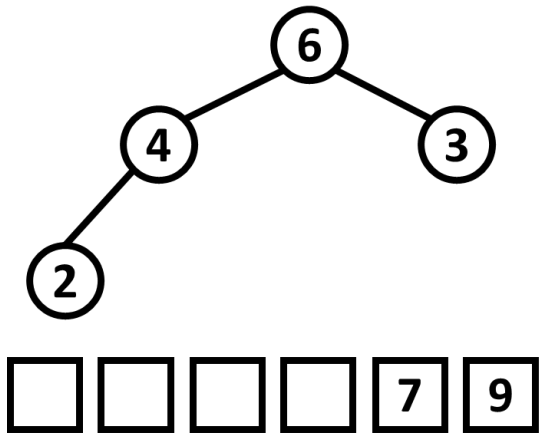
Ví dụ với mảng có các phần tử 9, 7, 6, 2, 4, 3. Hãy sắp xếp theo thứ tự tăng dần.

<p>Từ mảng ban đầu, tạo thành heap</p>	
<p>Lấy phần tử gốc ra khỏi Heap, đẩy phần tử cuối cùng của Heap lên làm nút gốc, thêm phần tử lấy ra vào cuối dãy sắp xếp.</p>	
<p>Heap sẽ được hiệu chỉnh lại cho đúng quy tắc. Nút cha lớn hơn 2 nút con.</p>	

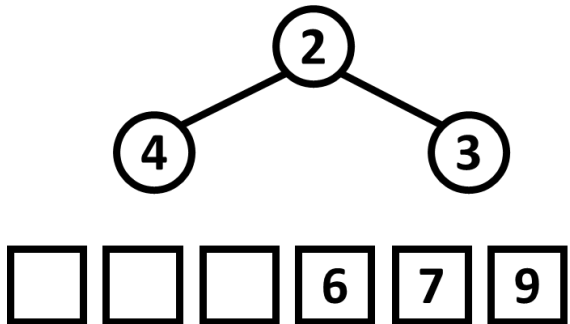
Tiếp tục, lấy nút gốc ra khỏi Heap, đưa phần tử lấy được vào dãy. Đưa phần tử cuối cùng lên làm nút gốc



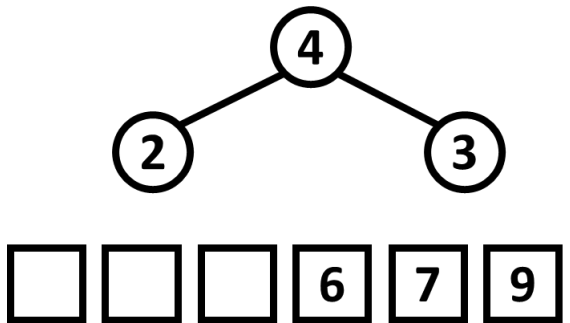
Hiệu chỉnh Heap

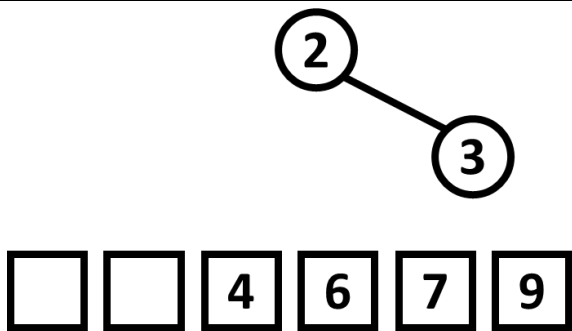
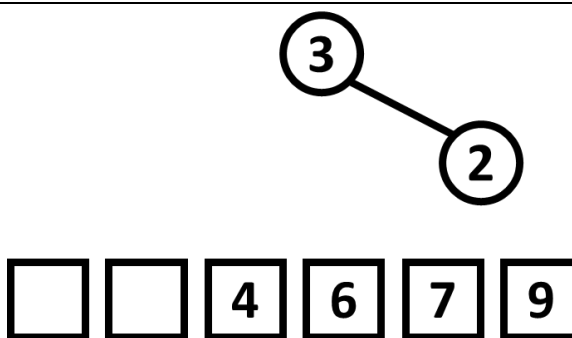
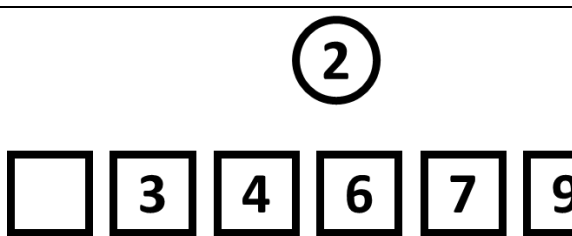



Lấy phần tử nút gốc là 6, đưa vào dãy, đưa phần tử cuối cùng lên làm nút gốc



Hiệu chỉnh Heap



Lấy phần tử nút gốc là 4, đưa vào dây, đưa phần tử cuối cùng lên làm nút gốc	
Hiệu chỉnh Heap	
Lấy phần tử nút gốc là 3 và đưa vào dây	
Đưa phần tử cuối cùng vào dây, ta được dãy sắp xếp tăng dần	

Bảng 2.6: Minh họa thuật toán sắp xếp vun đống

2.1.7 Đặc điểm của các thuật toán sắp xếp

Thuật toán sắp xếp	Độ phức tạp		Ưu điểm	Nhược điểm
	Tốt nhất	Tệ nhất		
Chèn	$O(n)$	$O(n^2)$	Tốt cho danh sách gần như đã sắp xếp.	Hiệu suất kém trên tập dữ liệu ngẫu nhiên lớn.
Chọn	$O(n^2)$	$O(n^2)$	Ít phép hoán đổi so với Bubble Sort.	Hiệu suất kém, không phù hợp cho dữ liệu lớn.
Trộn	$O(n \log n)$	$O(n \log n)$	Hiệu quả, ổn định, phù hợp cho dữ liệu lớn.	Sử dụng thêm bộ nhớ phụ, triển khai phức tạp hơn các thuật toán cơ bản.
Nhanh	$O(n \log n)$	$O(n^2)$	Hiệu quả, nhanh trong thực tế, không cần nhiều bộ nhớ phụ.	Tệ nhất nếu chọn pivot không tốt (có thể dùng

				Randomized Pivot để cải thiện).
Nổi bật	$O(n)$	$O(n^2)$	Đơn giản, dễ hiểu, dễ triển khai.	Hiệu suất kém trên tập dữ liệu lớn.
Vun đống	$O(n \log n)$	$O(n \log n)$	Không gian phụ thấp, luôn đạt hiệu suất tốt.	Không ổn định, phức tạp hơn Merge và Quick Sort trong triển khai.

Bảng 2.7: Đặc điểm của 6 thuật toán sắp xếp

2.2 Ngôn ngữ Python

2.2.1 Giới thiệu chung về Python

Python là ngôn ngữ lập trình hướng đối tượng, cấp cao, mạnh mẽ. Vào cuối những năm 1980, Guido Van Rossum là một lập trình viên người Hà Lan và ông mong muốn sử dụng một ngôn ngữ thông dịch như ABC (ABC có cú pháp rất dễ hiểu). Vì vậy, ông quyết định tạo ra một ngôn ngữ mở rộng. Điều này đã dẫn đến một thiết kế của ngôn ngữ mới, chính là Python sau này. Vào tháng 2 năm 1991, ngôn ngữ Python chính thức lần đầu ra mắt [9].



Hình 2.2: Cha đẻ của ngôn ngữ Python

Các phiên bản Python đã phát hành cho đến hiện tại

Thời gian	Phiên bản
01/1994	Python 1.0 (bản phát hành chuẩn đầu tiên)
05/09/2000	Python 1.6 (Phiên bản 1.x cuối cùng)
16/10/2000	Python 2.0 (Giới thiệu list comprehension)
03/07/2010	Python 2.7 (Phiên bản 2.x cuối cùng)
03/12/2008	Python 3.0 (Loại bỏ cấu trúc và mô-đun trùng lặp)
20/07/2020	Python 3.8.5 (Bổ sung thêm tính năng của hàm)
06/06/2024	Python 3.12.4 (Bản phát hành mới nhất)

Bảng 2.8: Các phiên bản Python

2.2.2 Đặc điểm của ngôn ngữ Python

Python có cú pháp rất đơn giản, rõ ràng. Nó dễ đọc và viết hơn rất nhiều khi so sánh với những ngôn ngữ lập trình khác như C++, Java, C#. Python làm cho việc lập trình trở nên thú vị, cho phép bạn tập trung vào những giải pháp chứ không phải cú pháp [9].

Các chương trình Python có thể di chuyển từ nền tảng này sang nền tảng khác và chạy nó mà không có bất kỳ thay đổi nào. Nó chạy liền mạch trên hầu hết tất cả các nền tảng như Windows, macOS, Linux [9].

Giả sử một ứng dụng đòi hỏi sự phức tạp rất lớn, bạn có thể dễ dàng kết hợp các phần code bằng C, C++ và những ngôn ngữ khác (có thể gọi được từ C) vào code Python. Điều này sẽ cung cấp cho ứng dụng của bạn những tính năng tốt hơn cũng như khả năng scripting mà những ngôn ngữ lập trình khác khó có thể làm được [9].

Không giống như C/C++, với Python, bạn không phải lo lắng những nhiệm vụ khó khăn như quản lý bộ nhớ, dọn dẹp những dữ liệu vô nghĩa,... Khi chạy code Python, nó sẽ tự động chuyển đổi code sang ngôn ngữ máy tính có thể hiểu. Bạn không cần lo lắng về bất kỳ hoạt động ở cấp thấp nào [9].

Python có một số lượng lớn thư viện tiêu chuẩn giúp cho công việc lập trình của bạn trở nên dễ thở hơn rất nhiều, đơn giản vì không phải tự viết tất cả code [9].

2.3 Thư viện Tinker

2.3.1 Giới thiệu chung

Tkinter là một thư viện trong ngôn ngữ lập trình Python được sử dụng để tạo giao diện đồ họa người dùng (GUI). "Tkinter" là viết tắt của "Tk interface", một toolkit đồ họa cung cấp các công cụ để phát triển giao diện người dùng.

Tkinter là một phần của thư viện tiêu chuẩn của Python và đã được tích hợp sẵn trong hầu hết các cài đặt Python. Điều này giúp cho Tkinter trở thành một lựa chọn phổ biến cho việc phát triển ứng dụng với giao diện đồ họa đơn giản trong Python.

Một số đặc điểm của Tkinter bao gồm khả năng tạo các thành phần giao diện như cửa sổ, nút, ô văn bản, và các widget khác để tương tác với người dùng. Tkinter cung cấp cả các sự kiện và phương thức để xử lý tương tác người dùng và thay đổi trạng thái của ứng dụng. [11].



Hình 2.3: Thư viện Tkinter

2.3.2 Các Widget trong Tkinter của Python

Widget trong Tkinter là các thành phần giao diện đồ họa (GUI) được sử dụng để xây dựng ứng dụng. Mỗi widget đại diện cho một phần tử giao diện, chẳng hạn như nút bấm, hộp văn bản, menu, thanh cuộn, nhãn, hoặc khung chứa.

Minh họa một số Widget: Label, Button, Entry, OptionMenu,...

```
from tkinter import *

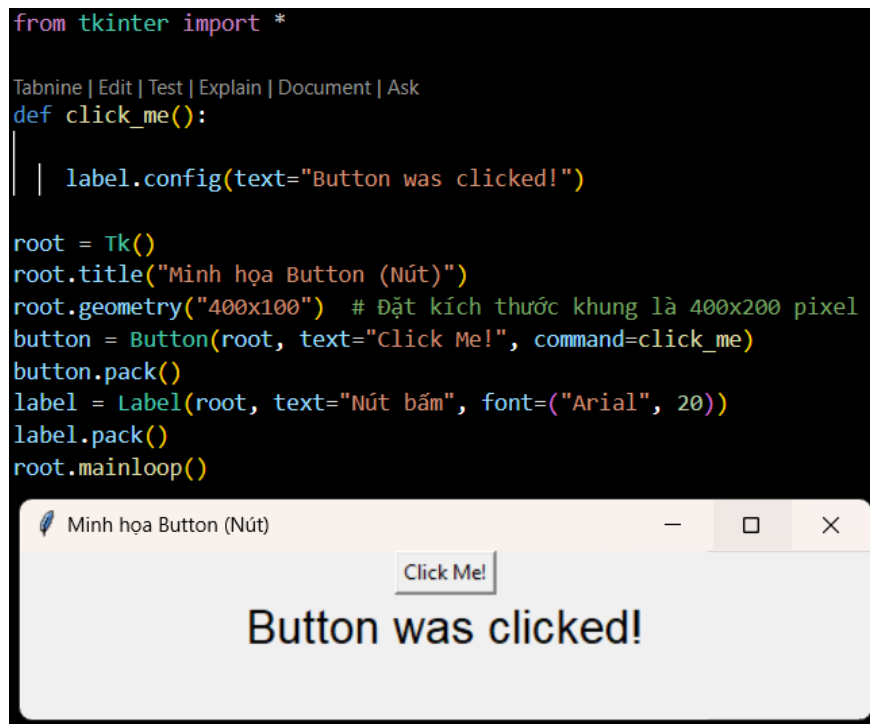
root = Tk()
root.title("Minh họa Label (Nhãn)")

root.geometry("400x100") # Đặt kích thước khung là 400x200 pixel

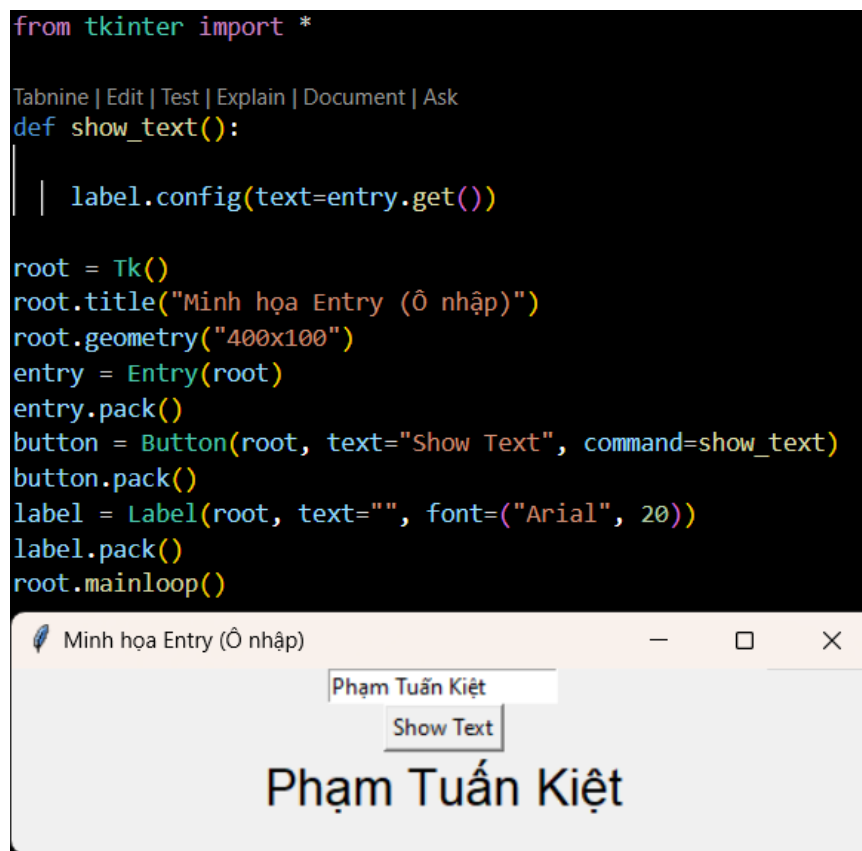
label = Label(root, text="Hello, Phạm Tuấn Kiệt", font=("Arial", 24))
label.pack()
root.mainloop()
```

The image shows a screenshot of a Tkinter window. The window has a title bar with the text "Minh họa Label (Nhãn)" and standard window control buttons (minimize, maximize, close). The main content area of the window displays the text "Hello, Phạm Tuấn Kiệt" in a large, bold, black font. The background of the window is light gray.

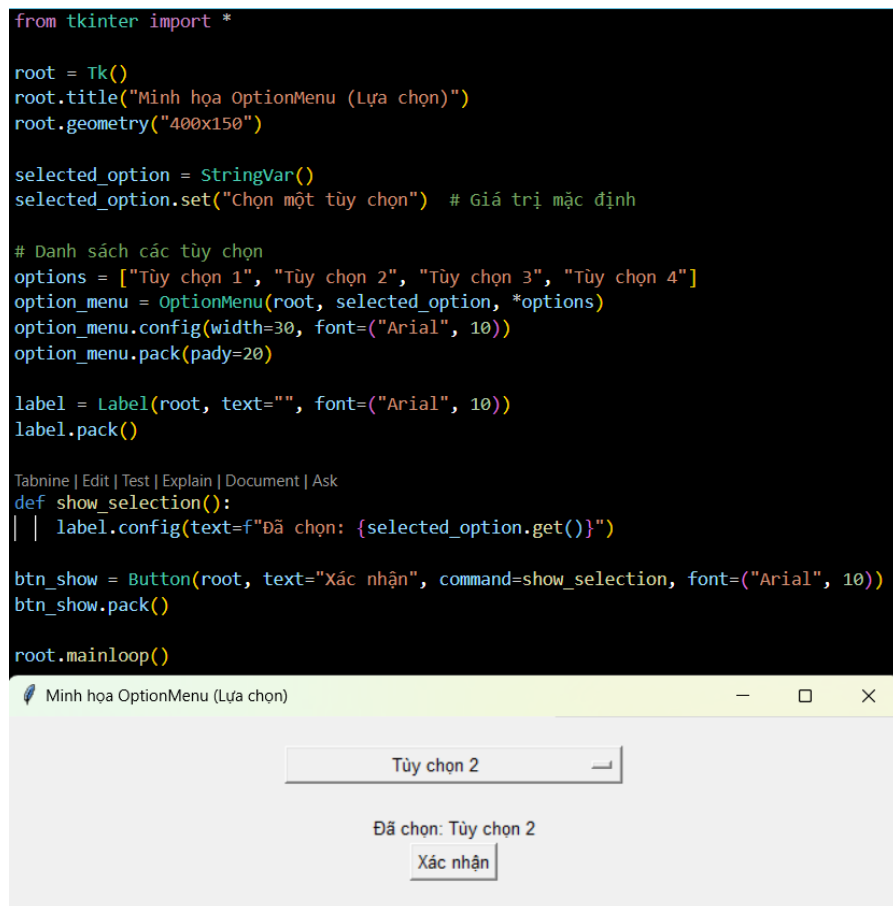
Hình 2.4: Minh họa Label trong Tkinter



Hình 2.5: Minh họa Button trong Tkinter



Hình 2.6: Minh họa Entry trong Tkinter



Hình 2.7: Minh họa OptionMenu trong Tkinter

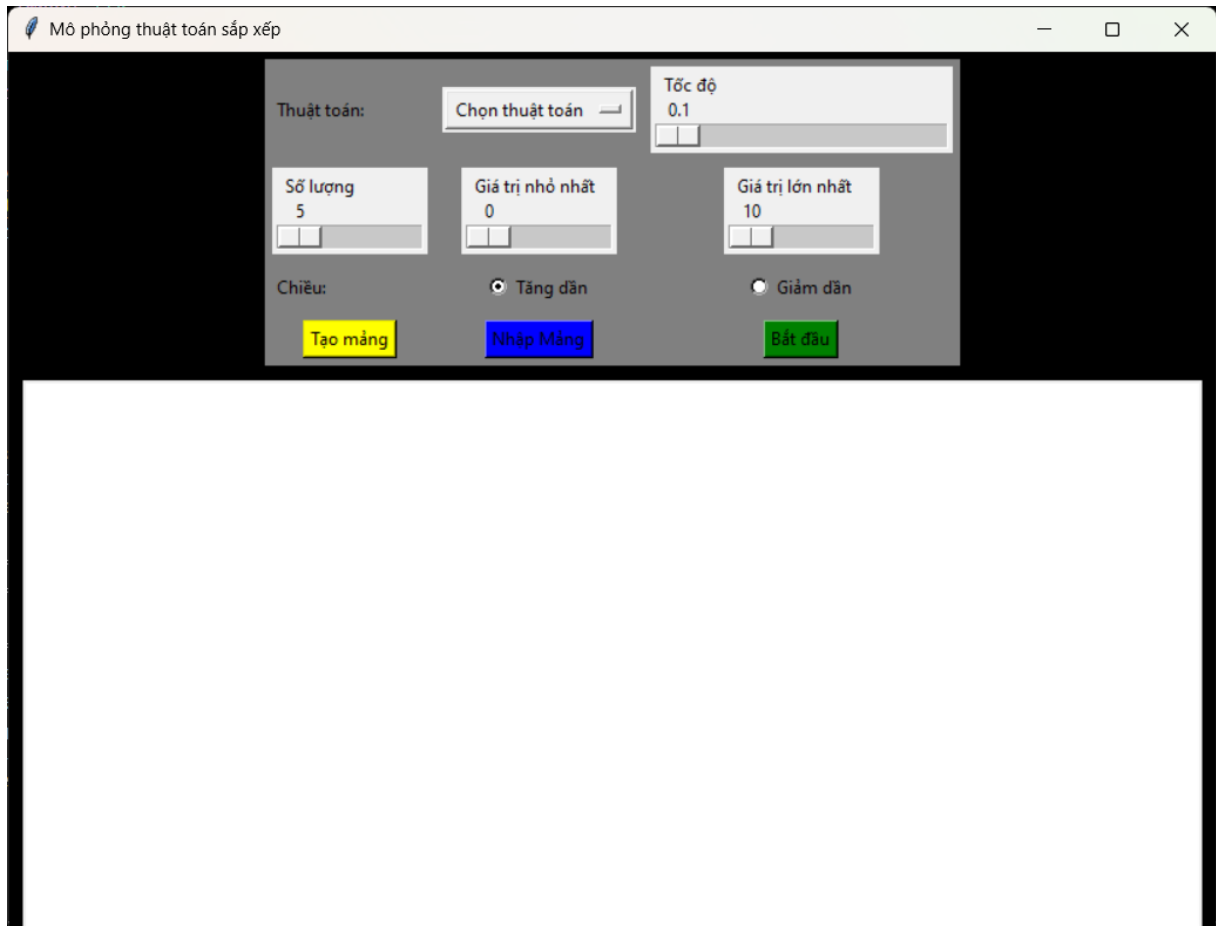


STT	Widget	Mô tả Chức Năng
1	Button	Thêm nút để tương tác với ứng dụng Python.
2	Canvas	Vẽ các đối tượng hoặc hình ảnh trên cửa sổ.
3	Checkbutton	Hiển thị và cho phép người dùng chọn hoặc bỏ chọn.
4	Entry	Hiển thị trường nhập dữ liệu một dòng cho người dùng.
5	Frame	Vùng chứa để nhóm hoặc tổ chức các widget khác.
6	Label	Hiển thị văn bản hoặc thông điệp cho người dùng.
7	ListBox	Hiển thị danh sách các tùy chọn hoặc mục.
8	Menubutton	Cung cấp một nút mở rộng để hiển thị danh sách menu.
9	Menu	Thêm các mục menu vào ứng dụng.
10	Message	Hiển thị tin nhắn hoặc thông báo cho người dùng.
11	Radiobutton	Cho phép người dùng chọn một trong nhiều tùy chọn.
12	Scale	Cung cấp thanh trượt cho người dùng để chọn giá trị trong một dải.
13	Scrollbar	Cung cấp thanh cuộn để người dùng có thể di chuyển trong nội dung.
14	Text	Cung cấp một vùng nhập văn bản nhiều dòng cho việc chỉnh sửa và hiển thị nội dung.
15	Toplevel	Tạo một cửa sổ phụ độc lập.
16	Spinbox	Cho phép người dùng chọn từ danh sách các giá trị.
17	PanedWindow	Chia một phần giao diện thành nhiều khu vực có thể thay đổi kích thước.
18	LabelFrame	Vùng chứa giúp nhóm các widget lại với nhau và cung cấp tiêu đề cho nhóm đó.

Hình 2.8: Một số Widget khác của Tkinter

2.3.3 Tạo giao diện cho người dùng tương tác

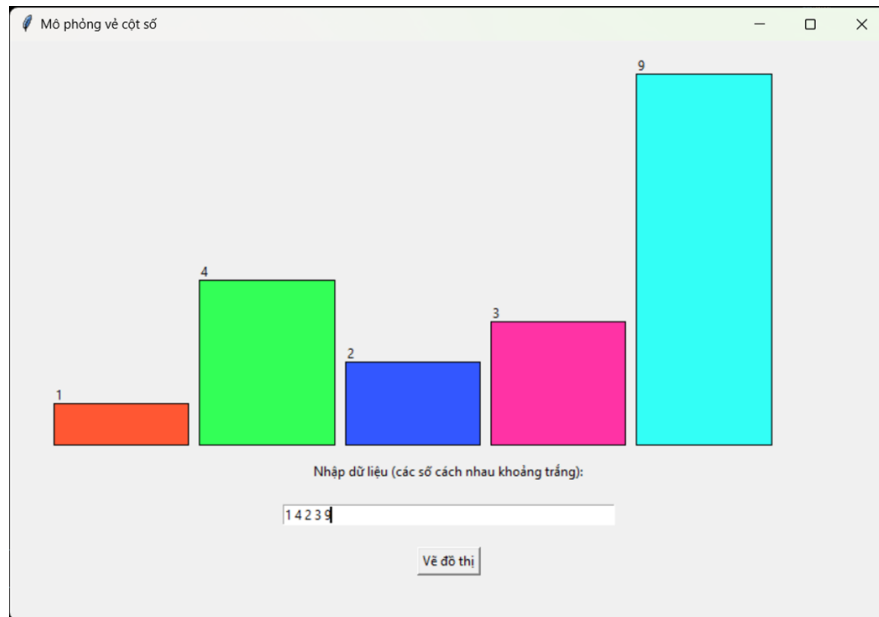
Sử dụng thư viện Tkinter cho phép người dùng thực hiện các thao tác nhập số lượng phần tử, giá trị của các phần tử và lựa chọn thuật toán mà người dùng muốn chương trình thực hiện. Đồng thời, ta thực hiện số điều kiện bắt buộc như số lượng phần tử phải thuộc một khoảng nào đó, các giá trị của phần tử phải nhập cho đúng với định dạng, hiển thị một số thông báo hướng dẫn người dùng.



Hình 2.9: Giao diện để tương tác với người dùng

2.3.4 Tạo các cột số

Các phần tử số được biểu diễn dưới dạng các cột số hình chữ nhật, được vẽ bằng canvas của thư viện Tkinter. Đây là một widget cho phép bạn vẽ hình vẽ (shapes), hình ảnh, và văn bản vào một vùng xác định trong cửa sổ. Những cột số này có độ rộng được tính bằng cách chia đều chiều rộng canvas cho số lượng phần tử + 1.



Hình 2.10: Mô phỏng các cột số

2.4 Các thư viện hỗ trợ khác

2.4.1 Thư viện Threading

Thư viện threading trong Python cung cấp các công cụ để thực thi song song các tác vụ trong các luồng (threads). Các luồng cho phép chạy nhiều phần của chương trình đồng thời, điều này giúp tăng hiệu suất, đặc biệt là khi cần xử lý các tác vụ I/O hoặc các tác vụ tốn thời gian.

```
import threading
import time

# Hàm đếm số chẵn
def count_even():
    for i in range(0, 20, 2): # Các số chẵn từ 0 đến 18
        print(f"Số chẵn: {i}")
        time.sleep(0.5) # Tạm dừng 0.5 giây

# Hàm đếm số lẻ
def count_odd():
    for i in range(1, 20, 2): # Các số lẻ từ 1 đến 19
        print(f"Số lẻ: {i}")
        time.sleep(0.5) # Tạm dừng 0.5 giây

# Tạo các luồng
thread_even = threading.Thread(target=count_even)
thread_odd = threading.Thread(target=count_odd)

# Bắt đầu các luồng
thread_even.start()
thread_odd.start()

# Đợi các luồng hoàn thành
thread_even.join()
thread_odd.join()

print("Hoàn thành!")
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS COMMENTS

Số lẻ: 3
Số chẵn: 4
Số lẻ: 5
Số chẵn: 6
Số lẻ: 7
Số chẵn: 8
Số lẻ: 9
Hoàn thành!
PS D:\TTCs\Visualize\Code>

Hình 2.11: Mô phỏng chạy đa luồng

Trong chương trình “Mô phỏng các thuật toán sắp xếp”, threading đóng vai trò đảm bảo thuật toán sắp xếp được chạy trong một luồng (thread) riêng biệt, giúp giao diện người dùng (GUI) vẫn mượt mà và không bị treo khi thuật toán sắp xếp đang được thực thi. Ví dụ như người dùng có thể tạm dừng thuật toán sắp xếp (bằng phím cách) mà không gặp vấn đề về giao diện bị treo (Not Responding).

2.4.2 Thư viện Time

Thư viện time trong Python là một thư viện tiêu chuẩn được sử dụng để làm việc với thời gian. Nó cung cấp các hàm để đo thời gian, chuyển đổi và định dạng thời gian. Với chương trình “Mô phỏng các thuật toán sắp xếp”, thư viện time cung cấp các lệnh giúp chương trình có các khoảng ngắt thời gian, giúp điều chỉnh tốc độ thực hiện của các thuật toán sắp xếp.

2.4.3 Thư viện Random

Thư viện random trong Python là một thư viện tiêu chuẩn được sử dụng để thực hiện các thao tác ngẫu nhiên. Nó cung cấp các hàm để tạo số ngẫu nhiên, chọn phần tử ngẫu nhiên từ danh sách, xáo trộn thứ tự, và thực hiện các phép tính ngẫu nhiên khác. Trong chương trình “Mô phỏng các thuật toán sắp xếp”, thư viện random giúp cho việc tạo ra một mảng số ngẫu nhiên dựa trên một khoảng giá trị mà người dùng quy định, giúp nhanh chóng thực hiện tạo ra một mảng số, rút ngắn quy trình chạy chương trình.

Chương III: PHÂN TÍCH THIẾT KẾ CHƯƠNG TRÌNH

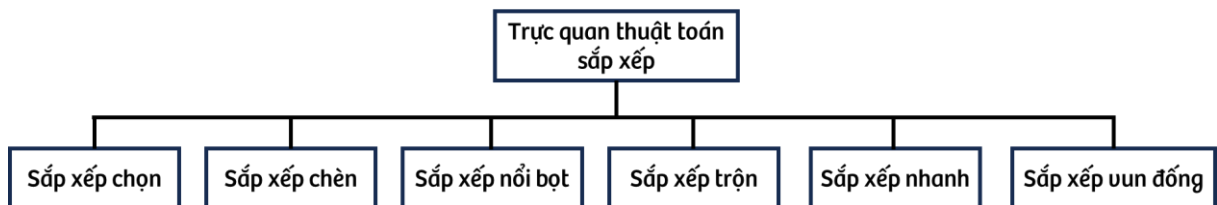
3.1 Mô tả hệ thống

“Mô phỏng các thuật toán sắp xếp bằng đồ họa” là một chương trình cho phép người dùng nhập vào một dãy số hữu hạn (30 phần tử), lựa chọn một trong 6 thuật toán (Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort) mong muốn. Chương trình sẽ thực hiện việc sắp xếp tương ứng với lựa chọn của người dùng. Sau cùng, người dùng sẽ thấy được quá trình các đối tượng di chuyển và sắp xếp, kết quả nhận được là một dãy số sẽ được sắp xếp.

3.2 Các yêu cầu cơ bản của hệ thống

Mô phỏng các thuật toán sắp xếp bằng đồ họa sẽ có:

- Chức năng sắp xếp chọn – Selection Sort
- Chức năng sắp xếp chèn – Insertion Sort
- Chức năng sắp xếp nổi bọt – Bubble Sort
- Chức năng sắp xếp trộn – Merge Sort
- Chức năng sắp xếp nhanh – Quick Sort
- Chức năng sắp xếp vun đống – Heap Sort



Hình 3.1: Sơ đồ phân rã chức năng

Chương IV: THIẾT KẾ CHƯƠNG TRÌNH

4.1 Tổng quan

Chương trình sẽ được chia nhỏ thành các module nhỏ để dễ dàng quản lý và sửa chữa khi cần thiết, mỗi module sẽ thực hiện các chức năng riêng. Có 2 module chính: Main.py và Sort.py

Sort.py là nơi chứa các lệnh, hàm thực tính toán, sắp xếp tương ứng với mỗi thuật toán riêng biệt, sử dụng các hàm riêng biệt tương ứng với mỗi thuật toán để diễn tả quá trình vẽ và di chuyển của các cột số.

Cuối cùng là Main.py, đây là nơi chứa các lệnh dùng để tương tác với người dùng là chính, tại đây sẽ yêu cầu người dùng nhập số lượng, giá trị các phần tử và chọn loại thuật toán mong muốn được thể hiện.

4.2 Module Sort

4.2.1 Hàm trực quan của sắp xếp chọn

Hàm `selection_sort` sẽ duyệt qua các phần tử trong mảng, sau đó tìm phần tử có giá trị nhỏ nhất trong mảng và đổi chỗ với phần tử đầu tiên. Mỗi lần thực hiện hàm sẽ gọi hàm `drawData` để thể hiện các bước hóa đổi của thuật toán sắp xếp chọn.

```
#####-SẮP XẾP CHỌN-#####
Tabnine | Edit | Test | Explain | Document
def selection_sort(data, drawData, timeTick, ascending=True, pause_flag=None):
    for i in range(len(data)-1):
        idx = i
        for j in range(i+1, len(data)):
            if (ascending and data[j] < data[idx]) or (not ascending and data[j] > data[idx]):
                idx = j
            if pause_flag:
                pause_flag.wait()

        if idx != i:
            data[i], data[idx] = data[idx], data[i]
            drawData(data, ['green' if x == i or x == idx else 'red' for x in range(len(data))])
            time.sleep(timeTick)
            if pause_flag:
                pause_flag.wait()

    drawData(data, ['green' for x in range(len(data))])
```

Hình 4.1: Hàm `selection_sort`

4.2.2 Hàm trực quan của sắp xếp chèn

Hàm `insertion_sort` duyệt qua các phần tử, bắt đầu từ phần tử thứ hai trong danh sách. Nếu phần tử hiện tại nhỏ hơn phần tử phía trước. Quá trình so sánh và hoán đổi này tiếp tục cho đến khi mảng được sắp xếp hoàn chỉnh. Sau khi thực hiện xong, hàm sẽ gọi `drawData` để vẽ lại mảng hoàn chỉnh.

```
#####-SẮP XẾP CHÈN-#####
Tabnine | Edit | Test | Explain | Document
def insertion_sort(data, drawData, timeTick, ascending=True, pause_flag=None):
    for i in range(1, len(data)):
        current_value = data[i]
        j = i - 1
        while j >= 0 and ((ascending and data[j] > current_value) or (not ascending and data[j] < current_value)):
            data[j + 1] = data[j]
            j -= 1
        if pause_flag:
            pause_flag.wait()
        data[j + 1] = current_value
        drawData(data, ['green' if x == j + 1 else 'red' for x in range(len(data))])
        time.sleep(timeTick)
        if pause_flag:
            pause_flag.wait()

    drawData(data, ['green' for x in range(len(data))])
```

Hình 4.2: Hàm insertion_sort

4.2.3 Hàm trực quan của sắp xếp nổi bọt

Hàm bubble_sort bắt đầu với phần tử đầu tiên và so sánh với phần tử phía bên phải. Nếu phần tử sau có giá trị nhỏ hơn, tiến hành đổi chỗ hai phần tử. Cứ như thế, liên tục đổi chỗ các phần tử cho đến khi mảng được sắp xếp theo thứ tự. Thực hiện gọi hàm drawData để hoàn chỉnh mảng số

```
#####-SẮP XẾP NỔI BỌT-#####
Tabnine | Edit | Test | Explain | Document
def bubble_sort(data, drawData, timeTick, ascending=True, pause_flag=None):
    for _ in range(len(data)-1):
        for j in range(len(data)-1):
            if (ascending and data[j] > data[j+1]) or (not ascending and data[j] < data[j+1]):
                data[j], data[j+1] = data[j+1], data[j]
                drawData(data, ['green' if x == j or x == j+1 else 'red' for x in range(len(data))])
                time.sleep(timeTick)
            if pause_flag:
                pause_flag.wait()

    drawData(data, ['green' for x in range(len(data))])
```

Hình 4.3: Hàm bubble_sort

4.2.4 Hàm trực quan của sắp xếp trộn

Thuật toán của sắp xếp trộn sẽ có 3 phần chính, chia mảng đầu vào thành các phần nhỏ hơn, sắp xếp từng phần và sau đó hợp nhất lại để tạo ra mảng đã sắp xếp

Hàm merge_sort thực hiện gọi đệ quy hàm merge_sort_alg để thực hiện chia nhỏ mảng số thành các phần nhỏ hơn.

```
#####-SẮP XẾP TRỘN-#####
Tabnine | Edit | Test | Explain | Document
def merge_sort(data, drawData, timeTick, ascending=True, pause_flag=None):
    merge_sort_alg(data, 0, len(data)-1, drawData, timeTick, ascending, pause_flag)
```

Hình 4.4: Hàm merge_sort

Hàm `merge_sort_alg` thực hiện chia nhỏ mảng số thành các mảng nhỏ hơn, mảng sẽ được chia làm 2 phần từ `left` → `middle` và từ `middle+1` → `right`. Sau đó, mảng sẽ được sắp xếp và gộp lại thông qua hàm `merge`.

```
Tabnine | Edit | Test | Explain | Document
def merge_sort_alg(data, left, right, drawData, timeTick, ascending, pause_flag):
    if left < right:
        middle = (left + right) // 2
        merge_sort_alg(data, left, middle, drawData, timeTick, ascending, pause_flag)
        merge_sort_alg(data, middle+1, right, drawData, timeTick, ascending, pause_flag)
        merge(data, left, middle, right, drawData, timeTick, ascending, pause_flag)
```

Hình 4.5: Hàm `merge_sort_alg`

Hàm `merge` thực hiện sắp xếp và hợp nhất các mảng con từ `left` → `middle` và `middle+1` → `right`. Các cột số sẽ được chia theo màu tương ứng với vị trí. Nếu cột số nằm trong khoảng `left` → `middle` sẽ có màu vàng, cột số nằm trong khoảng `middle+1` → `right` sẽ có màu hồng. Các cột số ngoài mảng đang sắp xếp sẽ có màu trắng.

```
Tabnine | Edit | Test | Explain | Document
def merge(data, left, middle, right, drawData, timeTick, ascending, pause_flag):
    if pause_flag:
        pause_flag.wait()
    drawData(data, getColorArray_merge(len(data), left, middle, right))
    time.sleep(timeTick)

    leftPart = data[left:middle+1]
    rightPart = data[middle+1:right+1]
    leftIdx, rightIdx = 0, 0

    for dataIndex in range(left, right+1):
        if leftIdx < len(leftPart) and rightIdx < len(rightPart):
            if (ascending and leftPart[leftIdx] <= rightPart[rightIdx]) or (not ascending and leftPart[leftIdx] >= rightPart[rightIdx]):
                data[dataIdx] = leftPart[leftIdx]
                leftIdx += 1
            else:
                data[dataIdx] = rightPart[rightIdx]
                rightIdx += 1
        elif leftIdx < len(leftPart):
            data[dataIdx] = leftPart[leftIdx]
            leftIdx += 1
        else:
            data[dataIdx] = rightPart[rightIdx]
            rightIdx += 1

    if pause_flag:
        pause_flag.wait()
    drawData(data, ["green" if x >= left and x <= right else "white" for x in range(len(data))])
    time.sleep(timeTick)
```

Hình 4.6: Hàm `merge`

4.2.5 Hàm trực quan của sắp xếp vun đống

Thuật toán sắp xếp vun đống sẽ 2 phần chính, hiệu chỉnh heap với mảng số ban đầu, thực hiện sắp xếp nhiều lần cho đến khi mảng được sắp xếp hoàn chỉnh

Hàm `heapfy` sẽ thực hiện công việc so sánh và hiệu chỉnh sao cho mảng đúng với tính chất nút cha luôn lớn hơn các nút con.

- Nút con bên trái: $2 \times i + 1$
- Nút con bên phải: $2 \times i + 2$

```
#####-SẮP XẾP VUN ĐỒNG-#####
Tabnine | Edit | Test | Explain | Document
def heapify(data, n, i, drawData, timeTick, ascending=True, pause_flag=None):
    largest_or_smallest = i
    left, right = 2*i + 1, 2*i + 2
    if left < n and ((ascending and data[left] > data[largest_or_smallest]) or (not ascending and data[left] < data[largest_or_smallest])):
        largest_or_smallest = left
    if right < n and ((ascending and data[right] > data[largest_or_smallest]) or (not ascending and data[right] < data[largest_or_smallest])):
        largest_or_smallest = right
    if largest_or_smallest != i:
        data[i], data[largest_or_smallest] = data[largest_or_smallest], data[i]
        drawData(data, ['green' if x == i or x == largest_or_smallest else 'red' for x in range(len(data))])
        time.sleep(timeTick)
    if pause_flag:
        pause_flag.wait()
    heapify(data, n, largest_or_smallest, drawData, timeTick, ascending, pause_flag)
```

Hình 4.7: Hàm heapfy

Hàm heap_sort sẽ thực hiện việc gọi hàm heapfy để tiến hành hiệu chỉnh cho mảng số. Sau đó, tiến hành trích xuất phần tử ra khỏi heap và hoán đổi nút gốc hiện tại với nút cuối. Cuối cùng, lại hiệu chỉnh heap mảng số đã giảm kích thước và gọi drawData để vẽ lại mảng sau mỗi bước

```
Tabnine | Edit | Test | Explain | Document
def heap_sort(data, drawData, timeTick, ascending=True, pause_flag=None):
    n = len(data)
    for i in range(n // 2 - 1, -1, -1):
        heapify(data, n, i, drawData, timeTick, ascending, pause_flag)
    for i in range(n - 1, 0, -1):
        data[i], data[0] = data[0], data[i]
        drawData(data, ['green' if x == i else 'red' for x in range(len(data))])
        time.sleep(timeTick)
        if pause_flag:
            pause_flag.wait()
        heapify(data, i, 0, drawData, timeTick, ascending, pause_flag)
    drawData(data, ['green' for x in range(len(data))])
```

Hình 4.8: Hàm heap_sort

4.2.6 Hàm trực quan của sắp xếp nhanh

Thuật toán sắp xếp nhanh có 2 phần chính. Đầu tiên, sẽ thực hiện chia mảng thành 2 phần nhỏ dựa trên phần tử pivot, so sánh các phần tử với pivot và tiến hành hoán đổi cho phù hợp. Cuối cùng, sẽ thực hiện đệ quy lại các bước trên để hoàn thành sắp xếp mảng và gọi drawData để tiến hành vẽ lại mảng sau mỗi bước.

Hàm partition sẽ tiến hành chia mảng thành 2 mảng con. Duyệt qua từ head đến tail-1, nếu phần tử hiện tại thỏa mãn điều kiện sắp xếp (nhỏ hơn pivot với sắp xếp tăng dần hoặc lớn hơn pivot với sắp xếp giảm dần), hoán đổi phần tử đó với phần tử ở vị trí border và tăng border. Cuối cùng, thực hiện gọi drawData để vẽ lại mảng.

```

Tabnine | Edit | Test | Explain | Document
def partition(data, head, tail, drawData, timeTick, ascending, pause_flag):
    border = head
    pivot = data[tail]
    drawData(data, getColorArray_quick(len(data), head, tail, border, border))
    time.sleep(timeTick)

    for j in range(head, tail):
        if pause_flag:
            pause_flag.wait()
        if (ascending and data[j] < pivot) or (not ascending and data[j] > pivot):
            drawData(data, getColorArray_quick(len(data), head, tail, border, j, True))
            time.sleep(timeTick)
            if pause_flag:
                pause_flag.wait()
            data[border], data[j] = data[j], data[border]
            border += 1

    data[border], data[tail] = data[tail], data[border]
    return border

```

Hình 4.9: Hàm partition

Hàm quick_sort sẽ tiến hành gọi đệ quy lại hàm partition cho phần mảng bên trái và bên phải cho đến khi mảng được sắp xếp. Phần tử pivot (tail) sẽ có màu xanh dương, phần tử tại vị trí phân hoạch border (head) có màu đỏ, phần tử đang được xét currIdx sẽ có màu vàng. Những phần tử nằm trong khoảng sang xét sẽ có màu xám, còn lại sẽ có màu trắng.

```

#####-SẮP XẾP NHANH-#####
Tabnine | Edit | Test | Explain | Document
def quick_sort(data, head, tail, drawData, timeTick, ascending, pause_flag=None):
    if head < tail:
        partitionIdx = partition(data, head, tail, drawData, timeTick, ascending, pause_flag)
        quick_sort(data, head, partitionIdx-1, drawData, timeTick, ascending, pause_flag)
        quick_sort(data, partitionIdx+1, tail, drawData, timeTick, ascending, pause_flag)

```

Hình 4.10: Hàm quick_sort

4.3 Module Main

4.3.1 Hàm hiển thị các cột số

Hàm Display sử dụng để trực quan hóa các cột số của mảng hiện tại đang xét. Đầu tiên, hàm sẽ xóa hết tất cả các phần tử trước đó đảm bảo cho khung hình hiển thị trạng thái mới nhất. Chiều cao của vùng hiển thị là $c_height = 370$ và chiều rộng là $c_width = 800$. Mỗi cột số có độ rộng được tính toán c_width đều cho kích thước của mảng cộng thêm 1, đảm bảo cho các cột số không bị đè lên nhau. Hàm sẽ thực hiện chuẩn hóa dữ liệu, đảm bảo cho các cột số có tỷ lệ chiều cao tương đối với nhau, đảm bảo cho giá trị lớn nhất sẽ có chiều cao luôn nằm trong cùng cho phép hiển thị.


```

Tabnine | Edit | Test | Explain | Document
def Display(data, colorArray):
    canvas.delete("all")
    c_height = 370
    c_width = 800
    x_width = c_width / (len(data) + 1)
    offset = 30
    spacing = 10
    normalizedData = [i / max(data) for i in data]
    for i, height in enumerate(normalizedData):
        x0 = i * x_width + offset + spacing
        y0 = c_height - height * 340
        x1 = (i + 1) * x_width + offset
        y1 = c_height
        canvas.create_rectangle(x0, y0, x1, y1, fill=colorArray[i])
        canvas.create_text(x0+2, y0, anchor=SW, text=str(data[i]))
    root.update_idletasks()

```

Hình 4.11: Hàm Display

4.3.2 Hàm tạo mảng số

Người dùng có thể tạo một mảng số ngẫu nhiên để thực hiện nhanh chóng việc trực quan hóa sắp xếp hoặc có thể nhập trực tiếp mảng số mong muốn.

```

Tabnine | Edit | Test | Explain | Document
def Create_Array():
    global data
    minVal = int(minEntry.get())
    maxVal = int(maxEntry.get())
    size = int(sizeEntry.get())
    data = [random.randrange(minVal, maxVal+1) for _ in range(size)]
    Display(data, ['blue' for _ in range(len(data))])

```

Hình 4.12: Hàm Create_Array tạo mảng số ngẫu nhiên

Hàm Input_Array sẽ hiển thị một khung của sổ cho phép nhập dữ liệu mảng mà người dùng mong muốn. Nếu dữ liệu trống hoặc sai định dạng thì chương trình sẽ hiển thị một cửa sổ thông báo cho người dùng biết.

```

Tabnine | Edit | Test | Explain | Document
def Input_Array():
    def confirmArray():
        global data
        input_text = inputEntry.get()
        # Chuyển đổi chuỗi nhập vào thành danh sách số nguyên
        data = list(map(int, input_text.split()))
        Display(data, ['blue' for _ in range(len(data))])
        inputWindow.destroy()

        if len(data) == 0:
            messagebox.showwarning("Cảnh báo", "Dữ liệu nhập vào không đúng")
        return

    # Tạo cửa sổ con
    inputWindow = Toplevel(root)
    inputWindow.title("Nhập Mảng")
    inputWindow.geometry("400x200")
    inputWindow.config(bg="white")

    Label(inputWindow, text="Nhập các phần tử của mảng (cách nhau bởi dấu cách):", bg="white").pack(pady=10)
    inputEntry = Entry(inputWindow, width=50)
    inputEntry.pack(pady=10)

    Button(inputWindow, text="Xác nhận", command=confirmArray, bg="green").pack(pady=10)

```

Hình 4.13: Hàm Input_Array cho phép nhập mảng mong muốn

4.3.3 Hàm lựa chọn thuật toán

Hàm Sort_Algorithm sẽ có nhiệm vụ nhận sự kiện của người dùng nhập vào và gọi đến hàm tương ứng của thuật toán sắp xếp.

```

Tabnine | Edit | Test | Explain | Document
def Sort_Algorithm():
    if selected_alg.get() == "Chọn thuật toán":
        messagebox.showwarning("Cảnh báo", "Hãy chọn một thuật toán!")
        return

    ascending = sort_order.get() == "Ascending"
    alg = selected_alg.get()

    # Định nghĩa hàm chạy thuật toán sắp xếp trong một thread
    def run_sort_algorithm():
        if alg == 'Quick Sort':
            quick_sort(data, 0, len(data)-1, Display, speedScale.get(), ascending, pause_flag)
        elif alg == 'Bubble Sort':
            bubble_sort(data, Display, speedScale.get(), ascending, pause_flag)
        elif alg == 'Merge Sort':
            merge_sort(data, Display, speedScale.get(), ascending, pause_flag)
        elif alg == 'Selection Sort':
            selection_sort(data, Display, speedScale.get(), ascending, pause_flag)
        elif alg == 'Insertion Sort':
            insertion_sort(data, Display, speedScale.get(), ascending, pause_flag)
        elif alg == 'Heap Sort':
            heap_sort(data, Display, speedScale.get(), ascending, pause_flag)

        # Sau khi thuật toán hoàn thành, hiển thị mảng đã sắp xếp
        Display(data, ['green' for _ in range(len(data))])

    # Tạo một thread mới để chạy thuật toán
    sort_thread = threading.Thread(target=run_sort_algorithm)
    sort_thread.start()

```

Hình 4.14: Hàm Sort_Algorithm

4.3.4 Hàm cho phép tạm dừng chương trình

Hàm `toggle_pause` cho phép nhận event của người dùng (SPACE) để chuyển đổi giữa trạng thái "tạm dừng" và "tiếp tục". Nếu `pause_flag` đang ở trạng thái "set" (chạy), thì `pause_flag.clear()` sẽ làm cho nó chuyển sang trạng thái "clear" (tạm dừng). Nếu `pause_flag` đang ở trạng thái "clear" (tạm dừng), thì `pause_flag.set()` sẽ làm cho nó trở lại trạng thái "set" (tiếp tục).

```
pause_flag = threading.Event()
pause_flag.set() # Bắt đầu ở trạng thái chạy

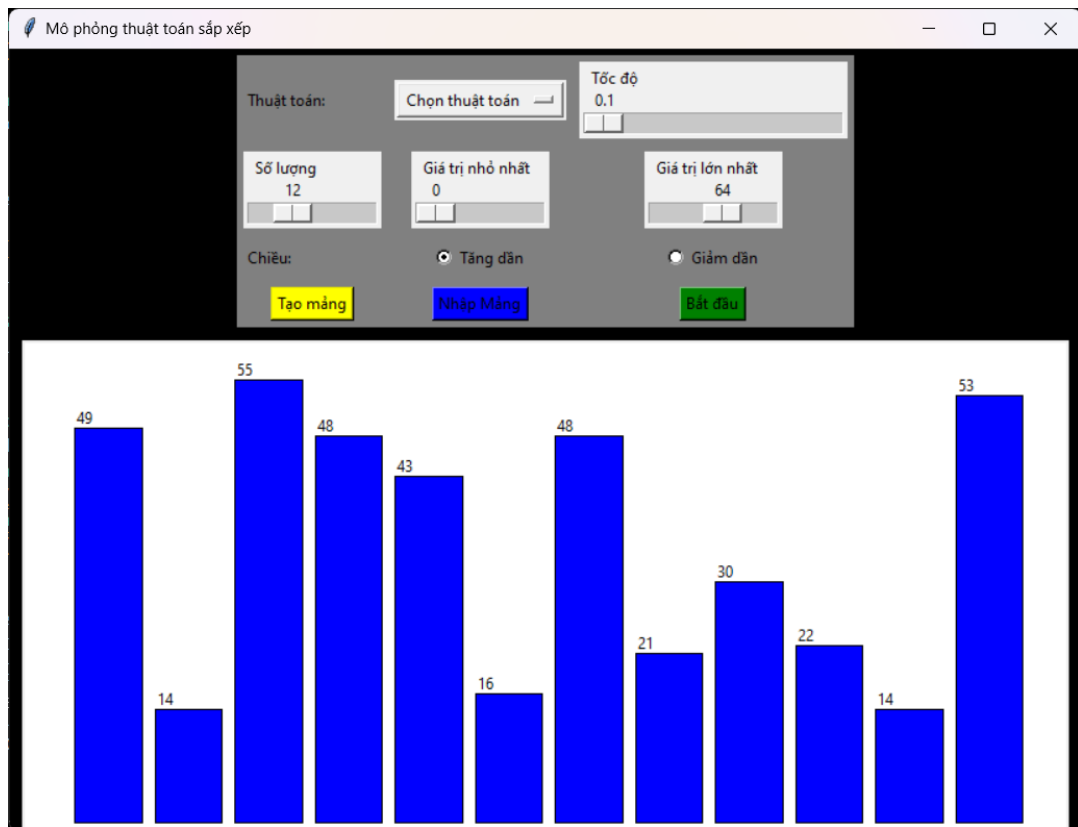
Tabnine | Edit | Test | Explain | Document
def toggle_pause(event=None):
    if pause_flag.is_set():
        pause_flag.clear() # Tạm dừng
    else:
        pause_flag.set() # Tiếp tục

while not pause_flag.is_set():
    root.update_idletasks() # Cho phép giao diện tiếp tục hoạt động
```

Hình 4.15: Hàm `toggle_pause`

4.3.5 Giao diện tương tác với người dùng

Giao diện người dùng sẽ được chia thành 2 phần chính. Phần trên `UI_frame` sẽ là vùng hiển thị các thao tác nhập các dữ liệu. Phần dưới canvas là vùng hiển thị các cột số, đây là nơi hiển thị cách các cột số được sắp xếp.



Hình 4.16: Giao diện tương tác chính của chương trình

Chương V: KẾT LUẬN

5.1 Ưu điểm – Kết quả đạt được

Trong quá trình thực hiện đề tài “Mô phỏng các thuật toán sắp xếp”, em đã đạt được một số yêu cầu cơ bản mà ban đầu đã đề ra, cụ thể:

- Tạo được chương trình mô phỏng 6 thuật toán sắp xếp.
- Các thuật toán có thể sắp xếp theo chiều tăng hoặc giảm mong muốn.
- Có giao diện tương tác cho người dùng có thể nhập mảng tùy ý, có thể lựa chọn thuật toán mong muốn, có thể tạo mảng số ngẫu nhiên hoặc nhập vào theo mong muốn,...
- Có thể tạm dừng chương trình tại các bước.

5.2 Nhược điểm – Hạn chế

Bên cạnh những yêu cầu đã đạt được, em cảm thấy chương trình còn một số hạn chế tồn đọng, cụ thể:

- Các đối tượng hiển thị chưa được đẹp mắt
- Chưa thực hiện được chức năng quay lại xem cái bước trước đó,...
- Kiến thức về các thư viện còn hạn chế, chưa áp dụng được nhiều,...

5.3 Kết luận

Trong quá trình thực hiện dự án, em đã có cơ hội áp dụng các kiến thức lý thuyết đã được học từ các học phần trên lớp vào thực tế, đặc biệt là trong việc lập trình cơ bản với ngôn ngữ Python. Đồng thời, em đã mở rộng hiểu biết của mình khi tiếp cận và sử dụng các thư viện quan trọng như **Tkinter**, **Random**, **Time** và **Threading**, giúp nâng cao khả năng lập trình và xử lý dữ liệu trực quan.

Chương trình em xây dựng đã cơ bản đáp ứng đúng các yêu cầu đề ra, thể hiện rõ cách hoạt động của các thuật toán sắp xếp theo thứ tự tăng dần hoặc giảm dần, đồng thời làm nổi bật các đặc trưng riêng biệt của từng thuật toán. Tuy nhiên, em nhận thấy chương trình vẫn còn một số hạn chế nhất định và cần được cải thiện thêm để hoàn thiện hơn trong tương lai.

TÀI LIỆU THAM KHẢO

- [1] GV. Trần Minh Văn (2023), *Cấu trúc dữ liệu và giải thuật – Sắp xếp mảng, Cây nhị phân*, Trường đại học Nha Trang
- [2] JD Hunter, 2007, "Matplotlib: A 2D Graphics Environment", *Computing in Science & Engineering*, tập 9, số 3, trang 90-95. doi:10.1109/mcse.2007.55
- [3] <https://viblo.asia/p/sap-xep-noi-bot-bubble-sort-la-gi-1VgZvN82ZAw> (viblo.asia, 19/3/2018, *Sắp xếp nổi bọt (Bubble Sort) là gì ?*) Truy cập 28/11/2024
- [4] <https://viblo.asia/p/sap-xep-chen-sap-xep-chon-va-sap-xep-tron-Do754zX4ZM6> (viblo.asia, 6/10/2021, *Sắp xếp chèn, Sắp xếp chọn và Sắp xếp trộn*) Truy cập 28/11/2024
- [5] <https://viblo.asia/p/sap-xep-vun-dong-63vKjeYk52R> (viblo.asia, 15/10/2021, *Sắp xếp vun đống*) Truy cập 28/11/2024
- [6] <https://fptshop.com.vn/tin-tuc/danh-gia/heap-sort-180656> (fptshop.com.vn, 5/2024, *Thuật toán Heap Sort là gì? Những kiến thức cần biết để ứng dụng thuật toán hiệu quả*) Truy cập 28/11/2024
- [7] <https://www.geeksforgeeks.org> (geeksforgeeks.org, *Insertion Sort Algorithm, Selection Sort Algorithm, Bubble Sort Algorithm, ...*) Truy cập 28/11/2024
- [8] <https://www.icantech.vn/kham-pha/tkinter> (icantech, 22/12/2023, *Tkinter Python là gì? Tất cả những gì bạn cần biết về Tkinter*) Truy cập 29/11/2024
- [9] <https://quantrimang.com/hoc/python-la-gi-tai-sao-nen-chon-python-140518> (quantrimang, 26/07/2024, *Python là gì? Tại sao nên chọn Python?*) Truy cập 29/11/2024
- [10] Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). *Khám phá cấu trúc, động lực và chức năng mạng lưới bằng cách sử dụng NetworkX*. Trong *Kỷ yếu Hội nghị Python trong Khoa học lần thứ 7 (SciPy 2008)* (tr. 11–15). Pasadena, CA, USA: SciPy.
- [11] <https://visualgo.net/en/sorting> *Sorting (Bubble, selection, insertion, merge, Quick, counting, Radix) - VisuAlgo*