

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

Giảng viên hướng dẫn: Nguyễn Mạnh Cường

Sinh viên thực hiện: Phạm Tuấn Kiệt

Mã số sinh viên: 64131060

Lớp 64.CNTT-4

Nha Trang – Khánh Hòa – Tháng 12/2024

MỤC LỤC

DANH MỤC HÌNH ẢNH	1
DANH MỤC BẢNG	3
Chương I: GIỚI THIỆU CHUNG	4
1. Lý do chọn đề tài	4
2. Mục tiêu đề tài	4
Chương II: CƠ SỞ LÝ THUYẾT	5
1. Các thuật toán sắp xếp	5
1.1 Sắp xếp chèn – Insertion Sort	5
1.2 Sắp xếp chọn – Selection Sort	6
1.3 Sắp xếp nổi bọt – Bubble Sort	8
1.4 Sắp xếp trộn – Merge Sort	9
1.5 Sắp xếp nhanh – Quick Sort	10
1.6 Sắp xếp vun đống – Heap Sort	11
1.7 Đặc điểm của các thuật toán sắp xếp	14
2. Ngôn ngữ Python	15
2.1 Giới thiệu chung về Python	15
2.2 Đặc điểm của ngôn ngữ Python	16
3. Thư viện Matplotlib	16
3.1 Giới thiệu chung	16
3.2 Phân cấp đối tượng trong Matplotlib	17
3.3 Module Pyplot của Matplotlib	17
3.4 Module Patches của Matplotlib	18
3.5 Mô phỏng hoạt họa của các đối tượng	18
4. Thư viện Networkx	19
4.1 Giới thiệu chung	19
4.2 Sử dụng Networkx vẽ cây nhị phân	20
5. Thư viện Tinker	21
5.1 Giới thiệu chung	21
5.2 Các Widget trong Tkinter của Python	22

5.3 Tạo giao diện cho người dùng tương tác.....	25
Chương III: PHÂN TÍCH THIẾT KẾ CHƯƠNG TRÌNH	26
1. Mô tả hệ thống.....	26
2. Các yêu cầu cơ bản của hệ thống	26
Chương IV: THIẾT KẾ CHƯƠNG TRÌNH	27
1. Tổng quan.....	27
2. Module Visualize.....	27
2.1 Hàm vẽ mảng số.....	27
2.2 Hàm hiệu ứng hoán đổi	28
2.3 Hàm hiệu ứng hợp nhất hai mảng con	30
2.4 Hàm vẽ cây nhị phân.....	31
2.5 Hàm di chuyển của các nút	33
2.6 Hàm hiệu chỉnh Heap.....	34
3. Module Sort.....	35
3.1 Hàm trực quan của sắp xếp chọn	35
3.2 Hàm trực quan của sắp xếp chèn.....	35
3.3 Hàm trực quan của sắp xếp nổi bọt.....	36
3.4 Hàm trực quan của sắp xếp trộn.....	36
3.5 Hàm trực quan của sắp xếp vun đống.....	37
3.6 Hàm phân hoạch cho thuật toán sắp xếp nhanh.....	37
3.7 Hàm trực quan của sắp xếp nhanh	38
4. Module Main	38
4.1 Hàm tương tác với người dùng	38
4.2 Giao diện tương tác với người dùng	39
Chương V: KẾT LUẬN	41
1. Ưu điểm – Kết quả đạt được	41
2. Nhược điểm – Hạn chế.....	41
3. Kết luận	41
TÀI LIỆU THAM KHẢO.....	42

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 Matplotlib	16
Hình 2.4: Minh họa phân cấp trong Matplotlib	17
Hình 2.5: Minh họa sử dụng Module Pylot	18
Hình 2.6: Minh họa sử dụng Module Patches.....	18
Hình 2.7: Mô phỏng di chuyển của các đối tượng.....	19
Hình 2.8: Thư viện Networkx	19
Hình 2.9: Mô phỏng cây nhị phân.....	20
Hình 2.10: Code tham khảo tạo cây nhị phân.....	21
Hình 2.11: Thư viện Tkinter.....	22
Hình 2.12: Minh họa Label trong Tkinter.....	22
Hình 2.13: Minh họa Button trong Tkinter	23
Hình 2.14: Minh họa OptionMenu trong Tkinter	24
Hình 2.15: Một số Widget khác của Tkinter.....	24
Hình 2.16: Giao diện để tương tác với người dùng	25
Hình 3.1: Sơ đồ phân rã chức năng.....	26
Hình 4.1: Hàm draw_array_as_squares	27
Hình 4.2: Mảng được vẽ bởi hàm draw_array_as_squares.....	28
Hình 4.3: Hàm di chuyển của các phần tử số.....	29
Hình 4.4: Các phần tử di chuyển bằng hàm swap_animation_... ..	30
Hình 4.5 Hàm merge hợp nhất hai mảng con	30
Hình 4.6: Minh họa hợp nhất hai mảng con	31
Hình 4.7: Hàm draw_binary_tree.....	32
Hình 4.8: Cây nhị phân vẽ bằng hàm draw_binary_tree	33
Hình 4.9: Hàm move_node	33
Hình 4.10: Hàm hiệu chỉnh heapify	34
Hình 4.11: Minh họa cây nhị phân đã hiệu chỉnh heap.....	35
Hình 4.12: Hàm selection_sort_visualize	35

Hình 4.13: Hàm <code>insertion_sort_visualize</code>	36
Hình 4.14: Hàm <code>bubble_sort_visualize</code>	36
Hình 4.15: Hàm <code>merge_sort_visualize</code>	37
Hình 4.16: Hàm <code>heap_sort_visualize</code>	37
Hình 4.17: Hàm <code>quick_sort_partition</code>	38
Hình 4.18: Hàm <code>quick_sort</code> và <code>quick_sort_visualize</code>	38
Hình 4.19: Hàm <code>interaction</code>	39
Hình 4.20: Giao diện tương tác chính của chương trình.....	40

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. 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.

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 các thư viện Matplotlib để có thể biểu diễn được chuyển động của các phần tử. Đòi kết hợp với Networkx để vẽ cây nhị phân và Tkinter để thiết kế giao diện người dùng

Chương II: CƠ SỞ LÝ THUYẾT

1. Các thuật toán sắp xếp

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

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

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 liên 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

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

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 phân 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	

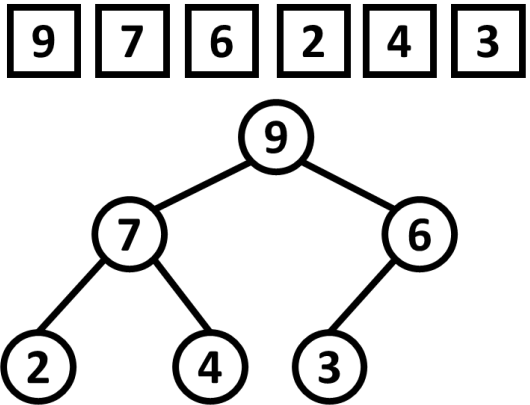
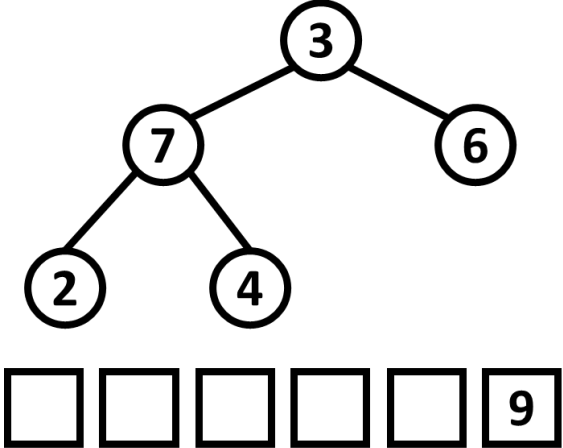
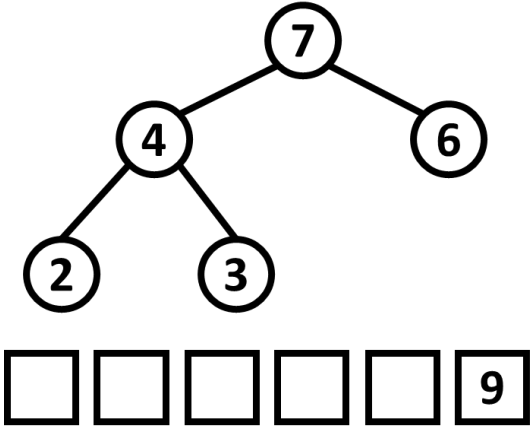
Bảng 2.5: Minh họa thuật toán sắp xếp nhanh

1.6 Sắp xếp vun đống – Heap Sort

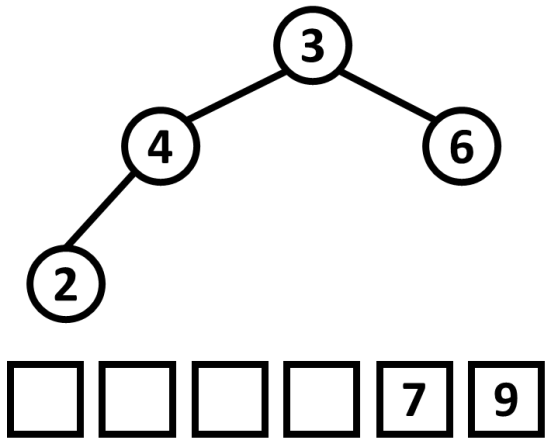
Heap là một cấu trúc dữ liệu dạng cây, trong đó các nút trên cây được sắp xếp theo một thứ tự ràng buộc nhất định giữa khóa của nút cha và khóa của nút con (thường là nút cha nhỏ hơn hoặc lớn hơn nút con). Nút ở gốc của Heap luôn luôn là nút có mức ưu tiên cao nhất, nghĩa là lớn nhất hoặc nhỏ nhất [5].

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].

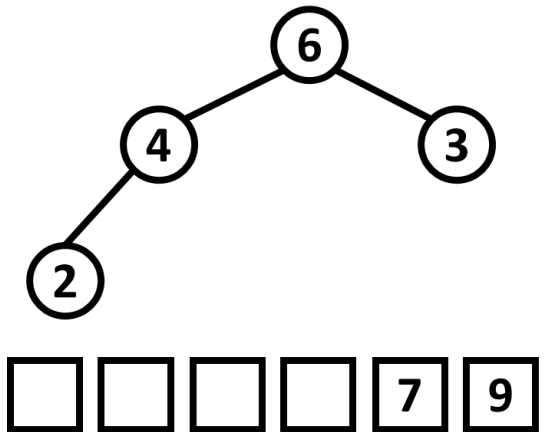
Ví dụ với mảng có các phần tử 9, 7, 6, 2, 4, 3. Hãy sắp xếp theo thứ tự tăng dần.

<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>	

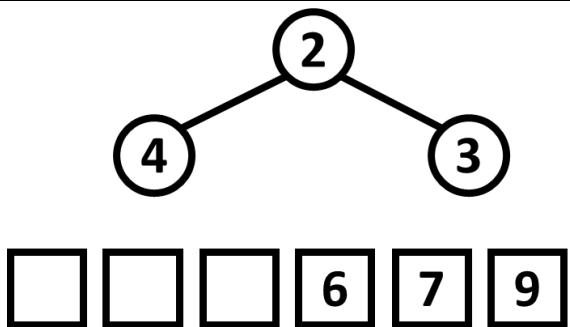
Tiếp tục, lấy nút gốc ra khỏi Heap, đưa phần tử lấy được vào dãy. Đưa phần tử cuối cùng lên làm nút gốc



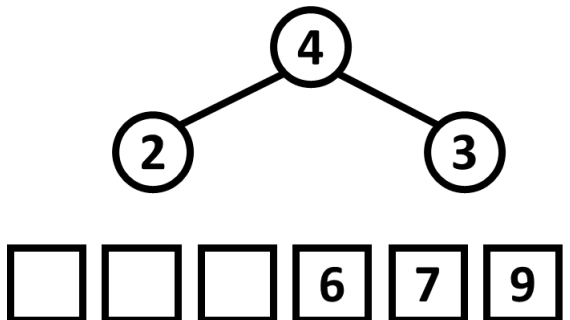
Hiệu chỉnh Heap

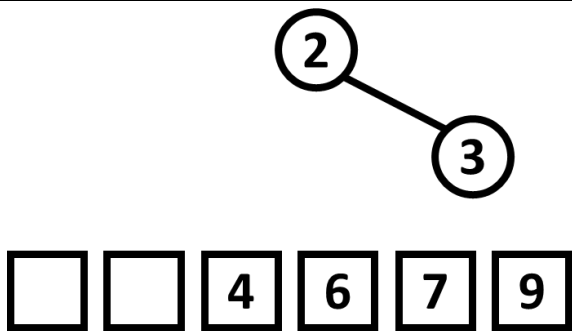
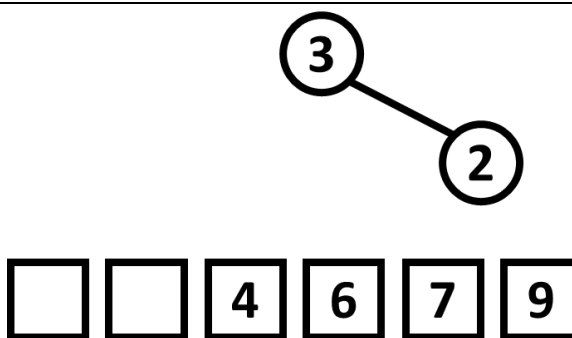
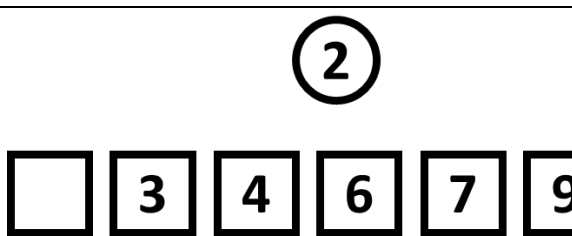



Lấy phần tử nút gốc là 6, đưa vào dãy, đưa phần tử cuối cùng lên làm nút gốc



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

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. Ngôn ngữ Python

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 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].

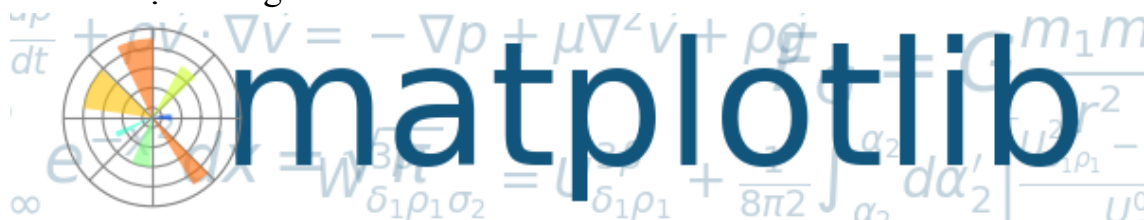
Giả sử một ứng dụng đòi hỏi sự phức tạp rất lớn, bạn có thể dễ dàng kết hợp các phần code bằng C, C++ và những ngôn ngữ khác (có thể gọi được từ C) vào code Python. Điều này sẽ cung cấp cho ứng dụng của bạn những tính năng tốt hơn cũng như khả năng scripting mà những ngôn ngữ lập trình khác khó có thể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

3. Thư viện Matplotlib

3.1 Giới thiệu chung



Hình 2.3: Thư viện Matplotlib

Matplotlib là một thư viện để tạo các biểu đồ 2D của mảng trong Python. Mặc dù nó có nguồn gốc từ việc mô phỏng các lệnh đồ họa MATLAB, nhưng nó độc lập với MATLAB và có thể được sử dụng theo cách hướng đối tượng, Pythonic. Mặc dù Matplotlib chủ yếu được viết bằng Python thuần túy, nhưng nó sử dụng nhiều NumPy và mã mở rộng khác để cung cấp hiệu suất tốt ngay cả đối với các mảng lớn [2]. Thư viện này thường được sử dụng trong phân tích dữ liệu, khoa học máy tính, học máy và nhiều lĩnh vực khác. Cùng với các thư viện hỗ trợ khác như NumPy và Pandas, Matplotlib trở thành công cụ không thể thiếu trong việc xử lý và trực quan hóa dữ liệu trong Python.

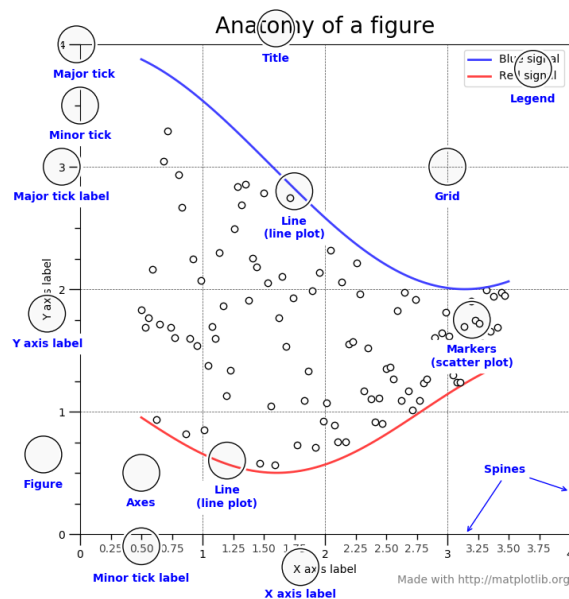
Tải thư viện:

- Mở terminal (Linux/macOS) hoặc CMD (Windows).
- Chạy lệnh “pip install matplotlib”

3.2 Phân cấp đối tượng trong Matplotlib

Một Matplotlib figure có thể được phân loại thành nhiều phần như dưới đây [8]:

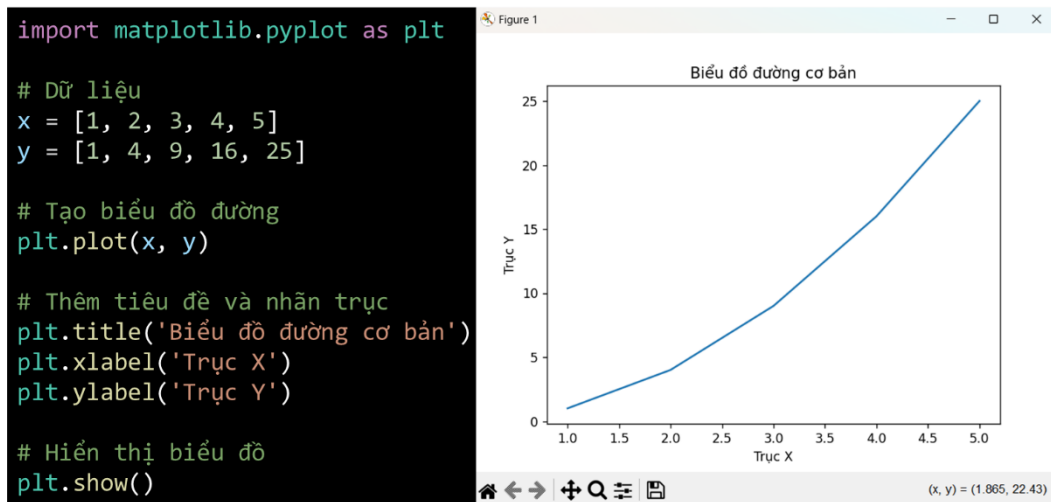
- **Figure:** Như một cái cửa sổ chứa tất cả những gì bạn sẽ vẽ trên đó.
- **Axes:** Thành phần chính của một figure là các axes (những khung nhỏ hơn để vẽ hình lên đó). Một figure có thể chứa một hoặc nhiều axes. Nói cách khác, figure chỉ là khung chứa, chính các axes mới thật sự là nơi các hình vẽ được vẽ lên.
- **Axis:** Chúng là dòng số giống như các đối tượng và đảm nhiệm việc tạo các giới hạn biểu đồ.
- **Artist:** Mọi thứ mà bạn có thể nhìn thấy trên figure là một artist như Text objects, Line2D objects, collection objects. Hầu hết các Artists được gắn với Axes.



Hình 2.4: Minh họa phân cấp trong Matplotlib

3.3 Module Pyplot của Matplotlib

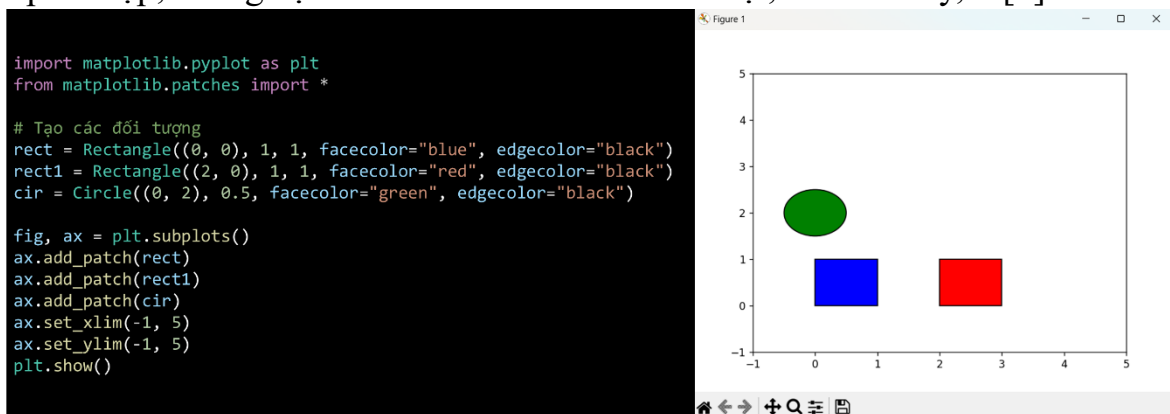
Pyplot là một module con của thư viện Matplotlib, được thiết kế để cung cấp một giao diện đơn giản và dễ sử dụng cho việc tạo đồ họa 2D. Pyplot cung cấp một tập hợp các hàm để tạo ra các loại đồ thị khác nhau, như đồ thị đường, biểu đồ cột và biểu đồ phân tán, theo cách tương tự như MATLAB, giúp người dùng thực hiện nhanh chóng các phép vẽ cần thiết [8].



Hình 2.5: Minh họa sử dụng Module Pylot

3.4 Module Patches của Matplotlib

Patches là một module của Matplotlib cung cấp các đối tượng hình học 2D như hình chữ nhật, hình tròn, hình đa giác. Patches giúp tạo các đối tượng hình học với các thuộc tính như màu sắc, độ trong suốt, viền,... Dễ dàng vẽ các hình học lên biểu đồ như biểu đồ phân tán hoặc biểu đồ hình tròn. Hỗ trợ tạo các hình vẽ phức tạp, chẳng hạn như các biểu đồ bản đồ nhiệt, biểu đồ cây,...[2]

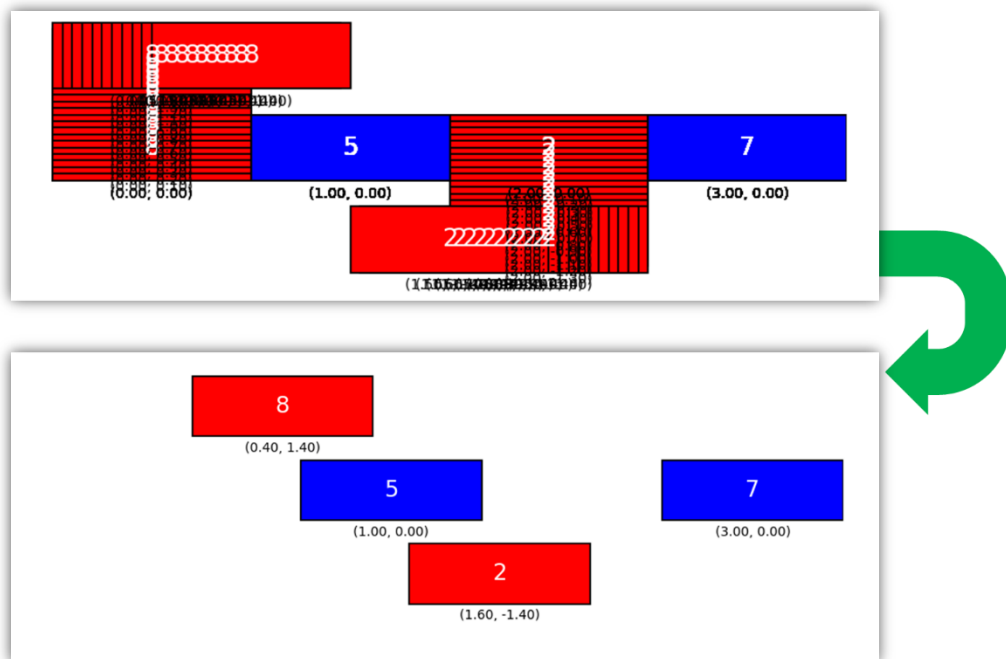


Hình 2.6: Minh họa sử dụng Module Patches

3.5 Mô phỏng hoạt hóa của các đối tượng

Để mô phỏng sự di chuyển của một đối tượng sử dụng matplotlib.patches, ta sẽ vẽ các đối tượng có kích thước cố định (1 đơn vị tọa độ). Để đối tượng có thể di chuyển, chúng ta cần biết vị trí hiện tại và vị trí đích đến của nó (vị trí tọa độ). Bằng cách tính toán khoảng cách giữa vị trí hiện tại và điểm đích, ta có thể cập nhật tọa độ của đối tượng qua từng bước di chuyển.

Quá trình di chuyển được mô phỏng bằng cách sử dụng vòng lặp để liên tục vẽ các hình từ vị trí hiện tại đến vị trí đích. Mỗi lần vẽ một hình mới, ta sẽ tính toán lại khoảng cách từ vị trí hiện tại đến điểm đích và xóa đi hình cũ ngay sau khi hình mới được vẽ. Quá trình này sẽ lặp lại cho đến khi đối tượng di chuyển hoàn toàn đến vị trí đích.



Hình 2.7: Mô phỏng di chuyển của các đối tượng

4. Thư viện Networkx

4.1 Giới thiệu chung

NetworkX là một thư viện của Python dùng để tạo, thao tác và nghiên cứu cấu trúc, động lực và chức năng của các mạng phức tạp. Nó cung cấp công cụ nghiên cứu cấu trúc và động lực của mạng lưới xã hội, sinh học và cơ sở hạ tầng. Một giao diện lập trình chuẩn và triển khai đồ thị phù hợp với nhiều ứng dụng và môi trường phát triển nhanh chóng cho các dự án hợp tác, đa ngành. Networkx hỗ trợ tăng tốc thuật toán và các tính năng bổ sung thông qua chương trình phụ trợ của bên thứ ba. Cung cấp một giao diện cho các thuật toán số hiện có và mã được viết bằng C, C++ và FORTRAN và khả năng làm việc dễ dàng với các tập dữ liệu lớn không chuẩn. Với NetworkX, bạn có thể tải và lưu trữ mạng theo định dạng dữ liệu chuẩn và không chuẩn, tạo nhiều loại mạng ngẫu nhiên và cổ điển, phân tích cấu trúc mạng, xây dựng mô hình mạng, thiết kế thuật toán mạng mới, vẽ mạng và nhiều tính năng khác [10].

Tải thư viện:

- Mở terminal (Linux/macOS) hoặc CMD (Windows).
- Chạy lệnh “pip install networkx”

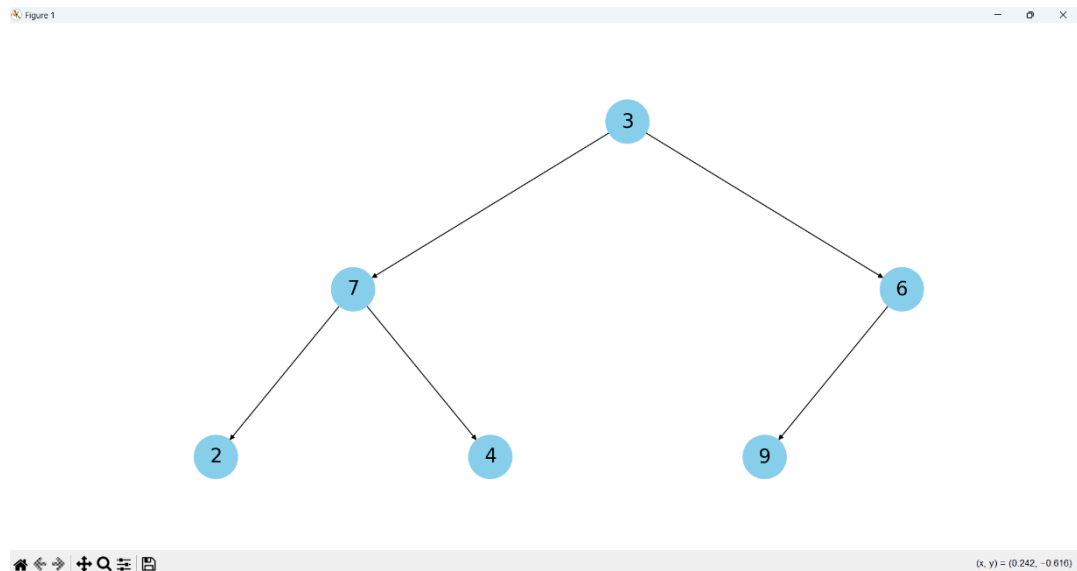


Hình 2.8: Thư viện Networkx

4.2 Sử dụng Networkx vẽ cây nhị phân

Sử dụng thư viện Networkx đồng thời kết hợp với thư viện Matplotlib để thực hiện chức năng vẽ một cây nhị phân từ một mảng đầu vào và hiển thị cây nhị phân dưới dạng đồ họa.

Ta xây dựng hàm “draw_binary_tree” nhận một mảng các giá trị, một đối tượng ax để vẽ cây và số lượng phần tử trong mảng. Đầu tiên, cây nhị phân được xây dựng dưới dạng đồ thị có hướng (DiGraph) với các nút và cạnh tương ứng. Mỗi phần tử trong mảng được gán cho một nút trong cây, và các cạnh được nối giữa các nút cha và con. Hàm đệ quy “add_edges_and_positions” được sử dụng để thêm các cạnh và xác định vị trí của các nút trong không gian hai chiều. Sau khi cây nhị phân được tạo, hàm “binary_tree_visualize” tạo một hình vẽ và gọi hàm “draw_binary_tree” để vẽ cây. Cuối cùng, “plt.show()” được sử dụng để hiển thị hình ảnh cây nhị phân. Kết quả là một đồ thị trực quan của cây nhị phân với các giá trị được phân bố theo cấu trúc cây nhị phân.



Hình 2.9: Mô phỏng cây nhị phân


```

import matplotlib.pyplot as plt
import networkx as nx

def draw_binary_tree(array, ax, n):
    ax.clear()
    G = nx.DiGraph()
    positions = {}
    edges = []

    def add_edges_and_positions(index, x, y, level_gap):
        if index >= n:
            return
        G.add_node(index, value=array[index])
        positions[index] = (x, y)
        left_child = 2 * index + 1
        right_child = 2 * index + 2
        if left_child < n:
            G.add_edge(index, left_child)
            edges.append((index, left_child))
            add_edges_and_positions(left_child, x - level_gap, y - 1, level_gap / 2)
        if right_child < n:
            G.add_edge(index, right_child)
            edges.append((index, right_child))
            add_edges_and_positions(right_child, x + level_gap, y - 1, level_gap / 2)

    add_edges_and_positions(0, 0, 0, 0.7)

    node_colors = ['skyblue' for _ in G.nodes()]
    nx.draw(G, pos=positions, ax=ax, with_labels=False, node_size=2000, node_color=node_colors,
    edgelist=edges)
    labels = {node: G.nodes[node]['value'] for node in G.nodes()}
    nx.draw_networkx_labels(G, pos=positions, labels=labels, ax=ax, font_size=20, font_color='black')

def binary_tree_visualize(array):
    fig, ax_tree = plt.subplots(figsize=(12, 8))
    n = len(array)

    draw_binary_tree(array, ax_tree, n)
    plt.show()

array = [3, 7, 6, 2, 4, 9]
binary_tree_visualize(array)

```

Hình 2.10: Code tham khảo tạo cây nhị phân

5. Thư viện Tinker

5.1 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 2.11: Thư viện Tkinter

5.2 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 *

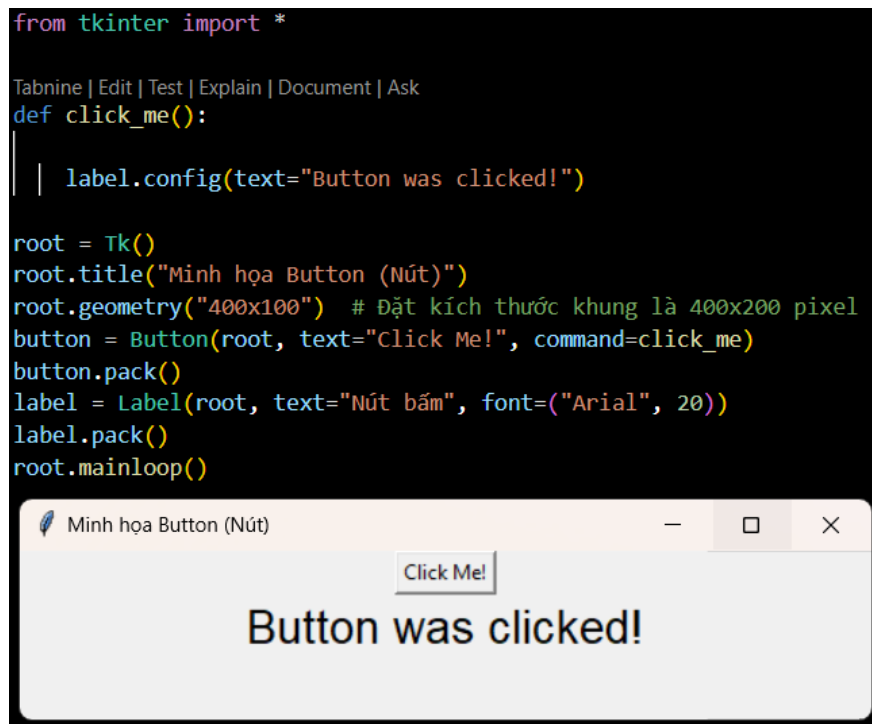
root = Tk()
root.title("Minh họa Label (Nhãn)")

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

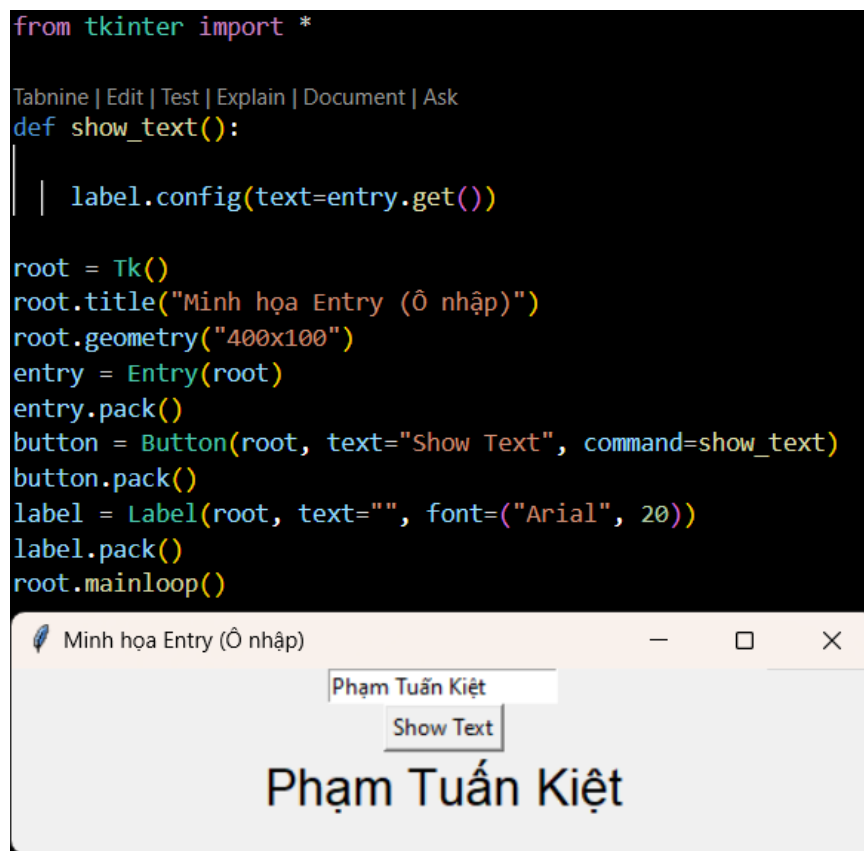
label = Label(root, text="Hello, Phạm Tuấn Kiệt", font=("Arial", 24))
label.pack()
root.mainloop()
```

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.

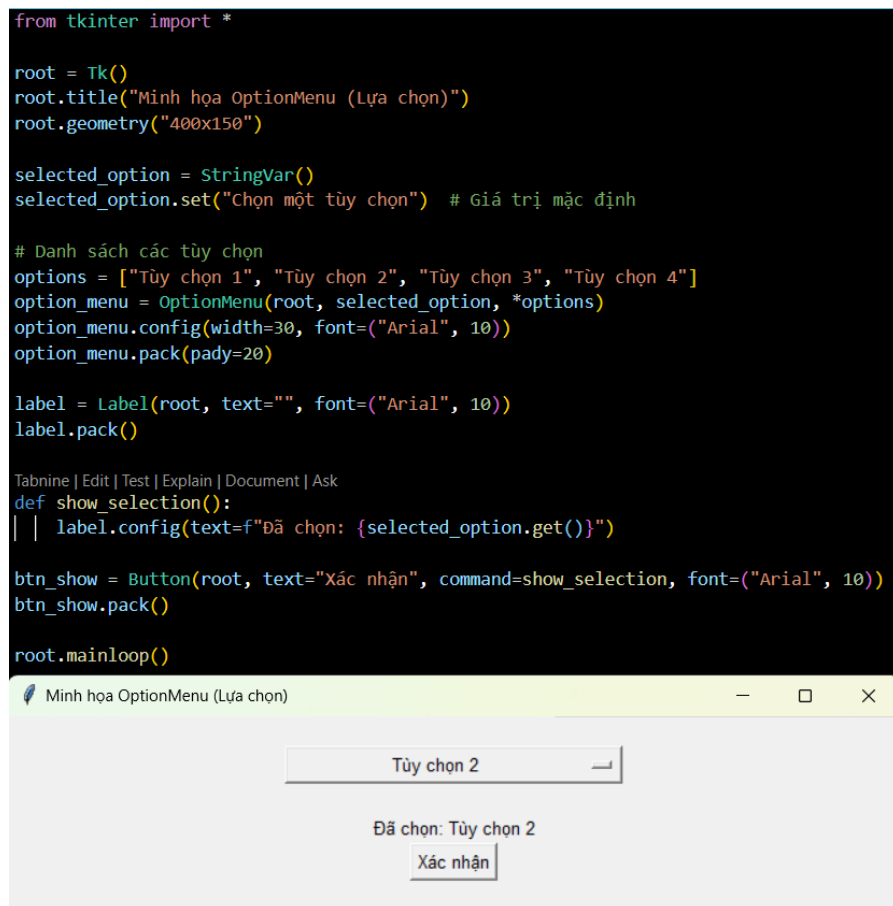
Hình 2.12: Minh họa Label trong Tkinter



Hình 2.13: Minh họa Button trong Tkinter



Hình 2.13: Minh họa Entry trong Tkinter



Hình 2.14: Minh họa OptionMenu trong Tkinter

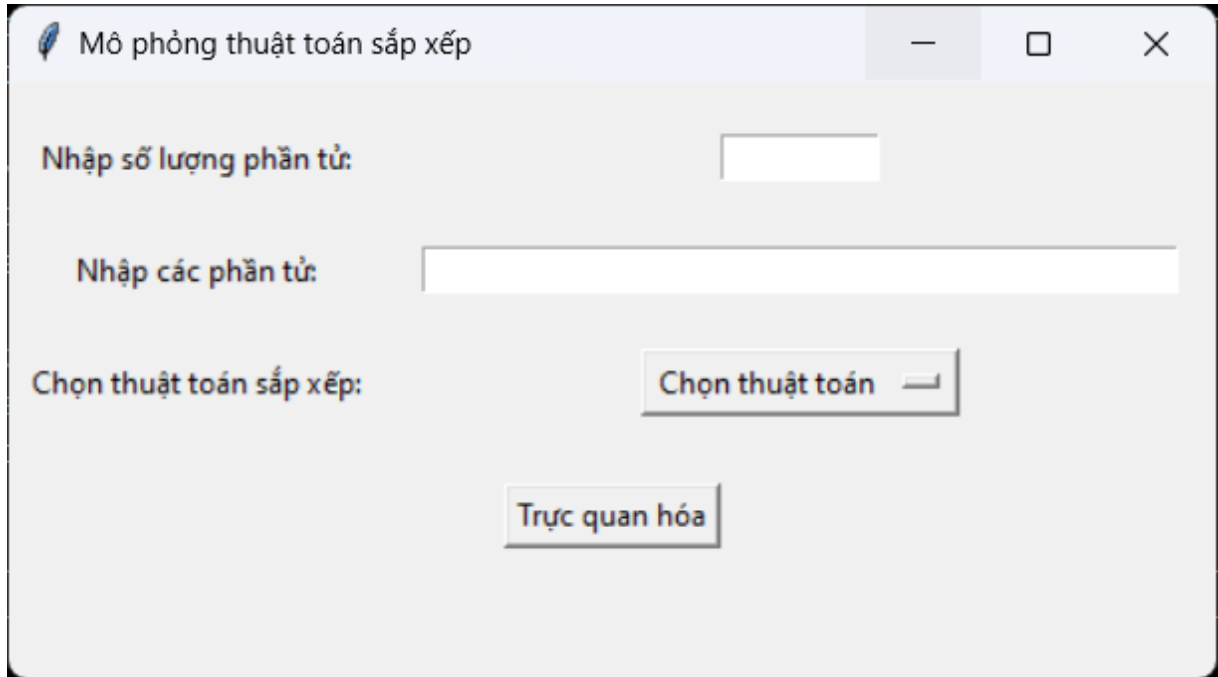


STT	Widget	Mô tả Chức Năng
1	Button	Thêm nút để tương tác với ứng dụng Python.
2	Canvas	Vẽ các đối tượng hoặc hình ảnh trên cửa sổ.
3	Checkbutton	Hiển thị và cho phép người dùng chọn hoặc bỏ chọn.
4	Entry	Hiển thị trường nhập dữ liệu một dòng cho người dùng.
5	Frame	Vùng chứa để nhóm hoặc tổ chức các 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 2.15: Một số Widget khác của Tkinter

5.3 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 2.16: Giao diện để tương tác với người dùng

Chương III: PHÂN TÍCH THIẾT KẾ CHƯƠNG TRÌNH

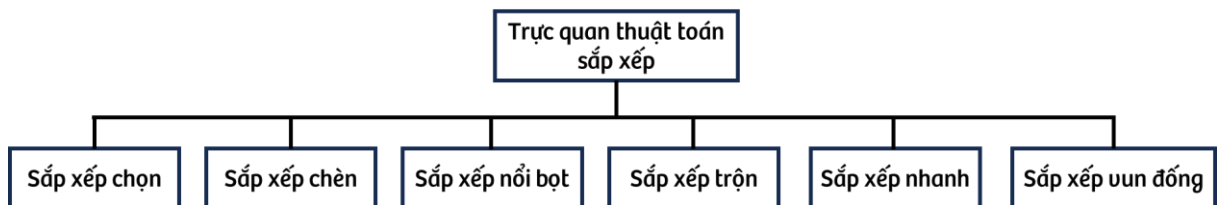
1. Mô tả hệ thống

“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 (20 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.

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

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 3 module chính, Visualize.py là module sẽ chứa các câu lệnh, hàm liên quan đến vẽ các ô số của mảng, nút số của cây nhị phân, vẽ cây nhị phân heap và các lệnh liên quan đến hiệu ứng di chuyển của các số. 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 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.

2. Module Visualize

2.1 Hàm vẽ mảng số

Hàm `draw_array_as_squares` sử dụng thư viện Matplotlib để vẽ các phần tử số trong mảng ở dạng hình vuông và chính giữa sẽ hiển thị giá trị tương ứng. Hàm này được sử dụng chung cho sắp xếp chọn, sắp xếp chèn, sắp xếp nổi bọt, sắp xếp nổi bọt và sắp xếp nhanh.

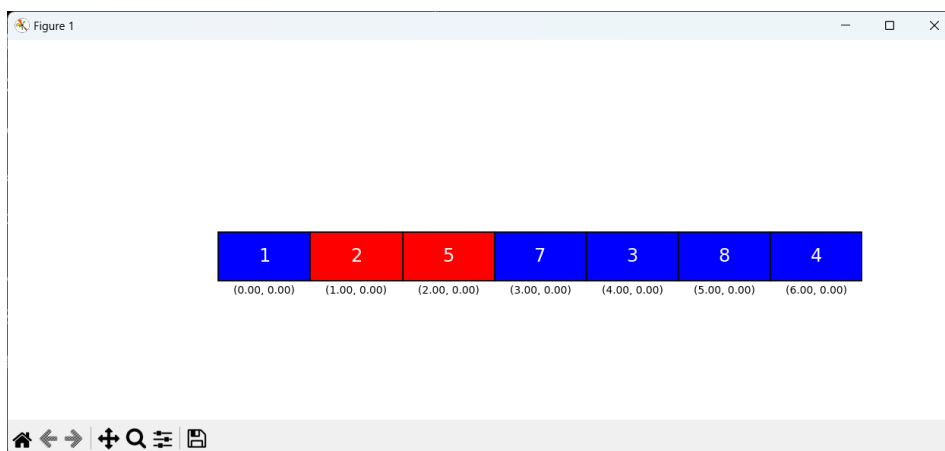
```
def draw_array_as_squares(array, highlight=None, ax=None, positions=None,
y_positions=None):
    ax.clear()
    ax.set_xlim(-1, len(array))
    ax.set_ylim(-2, 4)
    ax.axis('off') # Ẩn trục
    for i, value in enumerate(array):
        # Vẽ hình chữ nhật
        color = 'red' if highlight and i in highlight else 'blue'
        rect = patches.Rectangle((positions[i], y_positions[i]), 1, 1,
edgecolor='black', facecolor=color)
        ax.add_patch(rect)
        # Vẽ giá trị bên trong ô
        ax.text(positions[i] + 0.5, y_positions[i] + 0.5, str(value),
ha='center', va='center', fontsize=14, color='white')
```

Hình 4.1: Hàm `draw_array_as_squares`

Hàm `draw_array_as_squares` có các tham số đầu vào

- Mảng số `array` chứa các giá trị cần vẽ.
- Danh sách các chỉ số (index) trong mảng cần được làm nổi bật highlight (mặc định là `None`).
- Đối tượng Axes – `ax` của Matplotlib, dùng để vẽ các hình chữ nhật.
- Danh sách `positions` chứa tọa độ x của các ô vuông.
- Đối với sắp xếp nhanh, có thêm tham số là phần tử chốt pivot.

Cách hoạt động của hàm `draw_array_as_squares`: Đầu tiên, sử dụng hàm `ax.clear()` xóa nội dung của biểu đồ hiện tại để đảm bảo không bị chồng lấp với các lần vẽ trước. Thiết lập giới hạn trục x và y bằng `ax.set_xlim()` và `ax.set_ylim()` để đảm bảo hiển thị đầy đủ mảng số trên màn hình. Bên cạnh đó, ta ẩn trục của biểu đồ đi `ax.axis('off')` và chỉ tập trung hiển thị mảng số. Sử dụng vòng lặp `for` duyệt qua tất cả các phần tử và vị trí của phần tử đó trong mảng, nếu phần tử tại vị trí `i` nằm trong danh sách cần được highlight thì sẽ có màu đỏ, nếu không thì mặc định là màu xanh dương. Mỗi phần tử sẽ được tạo thành một hình chữ nhật tại vị trí (`positions[i]`, `y_positions[i]`) với kích thước `1x1`, mỗi hình có đường viền màu đen và bề mặt của màu sẽ phụ thuộc tương ứng với highlight. Các hình chữ nhật này sẽ được thêm vào biểu đồ bằng `ax.add_patch()`. Cuối cùng, hiển thị các giá trị tương ứng vào trung tâm mỗi ô với kích thước là 14 và chữ có màu là trắng.



Hình 4.2: Mảng được vẽ bởi hàm `draw_array_as_squares`

2.2 Hàm hiệu ứng hoán đổi

Các thuật toán sắp xếp chọn, sắp xếp chèn, sắp xếp nổi bọt và sắp xếp nhanh nhìn chung đều sử dụng cơ chế hoán đổi vị trí của các phần tử với nhau, sao cho các phần tử sau một loạt các thao tác hoán đổi thì phần tử đó sẽ ở vị trí đúng theo thứ tự.

Mặc dù các hàm có chút khác biệt, nhưng các hàm đều có 4 bước chung một cách thức hoạt động.

```

def swap_animation____ (array, i, j, ax):
    n_frames = 30 # Số khung hình cho hoạt ảnh
    positions = list(range(len(array)))
    y_positions = [0] * len(array)

    # Bước 1: Di chuyển lên/xuống (trục y) để tránh đè lên nhau
    for frame in range(n_frames // 2):
        y_positions[i] = frame / 10 # Di chuyển ô i lên
        y_positions[j] = -frame / 10 # Di chuyển ô j xuống
        draw_array_as_squares(array, highlight=[i, j], ax=ax, positions=positions, y_positions=y_positions)
        plt.pause(0.05)

    # Bước 2: Di chuyển ngang (trục x) để đổi chỗ
    x_distance = positions[j] - positions[i] # Khoảng cách trên trục x
    step = x_distance / (n_frames // 2) # Khoảng cách di chuyển mỗi khung hình
    for frame in range(n_frames // 2):
        positions[i] += step
        positions[j] -= step
        draw_array_as_squares(array, highlight=[i, j], ax=ax, positions=positions, y_positions=y_positions)
        plt.pause(0.05)

    # Bước 3: Trở về trục X (y = 0)
    for frame in range(n_frames // 2):
        y_positions[i] = max(0, y_positions[i] - 1 / 15) # Di chuyển dần xuống
        y_positions[j] = min(0, y_positions[j] + 1 / 15) # Di chuyển dần lên
        draw_array_as_squares(array, highlight=[i, j], ax=ax, positions=positions, y_positions=y_positions)
        plt.pause(0.05)

    # Bước 4: Cập nhật giá trị hoán đổi trong mảng
    array[i], array[j] = array[j], array[i]

```

Hình 4.3: Hàm di chuyển của các phần tử số

Hàm `swap_animation_....` có các tham số:

- Mảng array chứa các giá trị cần hoán đổi và tạo hoạt ảnh.
- Chỉ số của phần tử i và phần tử j muốn hoán đổi.
- Đối tượng Axes của Matplotlib dùng để vẽ hoạt ảnh.
- Đối với sắp xếp nhanh, có thêm tham số là phần tử chốt pivot.

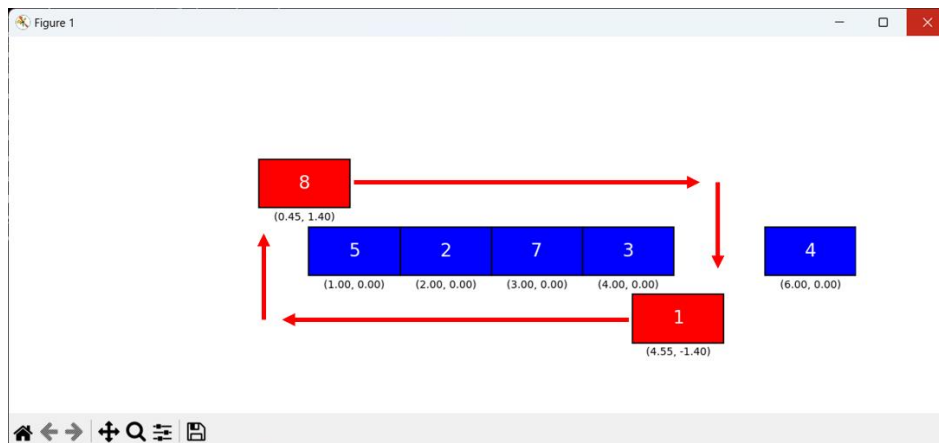
Cách thức hoạt động của hàm `swap_animation_...`: Đầu tiên, ta sẽ khởi tạo các giá trị như số khung hình cho hoạt ảnh `n_frames`, danh sách tạo độ x và tọa độ y mặc định bằng 0. Sau đó, lần lượt thực hiện 4 bước sau:

Bước 1, ta sử dụng vòng lặp để thiết lập vị trí di chuyển lên và xuống của các phần tử, phần tử i di chuyển dần lên trên, phần tử j di chuyển dần xuống dưới.

Bước 2, ta tính khoảng cách giữa 2 phần tử `x_distance` và chia đều mỗi bước di chuyển trên mỗi khung hình `step`. Ta sử dụng vòng lặp để di chuyển phần tử i sang bên phải, phần tử j sang bên trái cho tới khi đúng vị trí.

Bước 3, ta đưa các phần tử về vị trí ban đầu tại ($y = 0$), sử dụng vòng lặp để phần tử i di chuyển dần xuống, phần tử j di chuyển dần lên trở về trục ngang ($y = 0$).

Bước 4, sau khi các phần tử đã về đúng vị trí, ta cập nhật lại vị trí i và j của hai phần tử đó trong mảng.



Hình 4.4: Các phần tử di chuyển bằng hàm `swap_animation`...

2.3 Hàm hiệu ứng hợp nhất hai mảng con

Hàm hiệu ứng hợp nhất hai mảng con của thuật toán sắp xếp chọn, hàm này dùng để biểu diễn quá trình phân tách mảng thành các mảng con rồi lần lượt sắp xếp chúng theo thứ tự, quá trình này thực hiện cho đến khi mảng được sắp xếp.

```
def merge(array, start, mid, end, ax, positions, y_positions, order="ascending"):
    left = array[start:mid + 1]
    right = array[mid + 1:end + 1]
    l_idx = 0
    r_idx = 0
    sorted_idx = start
    # Tăng vị trí y của mảng con để hiển thị rõ các bước hợp nhất
    for i in range(start, end + 1):
        y_positions[i] -= 1

    draw_array_as_squares(array, ax=ax, positions=positions, y_positions=y_positions)
    plt.pause(0.5)

    # Hợp nhất hai mảng con
    while l_idx < len(left) and r_idx < len(right):
        if (order == "ascending" and left[l_idx] <= right[r_idx]) or \
            (order == "descending" and left[l_idx] >= right[r_idx]):
            array[sorted_idx] = left[l_idx]
            highlight = [sorted_idx]
            l_idx += 1
        else:
            array[sorted_idx] = right[r_idx]
            highlight = [sorted_idx]
            r_idx += 1

        # Cập nhật y_positions cho cả hai phần tử đang so sánh
        y_positions[sorted_idx] += 1
        draw_array_as_squares(array, ax=ax, highlight=highlight, positions=positions, y_positions=y_positions)
        plt.pause(0.5)
        sorted_idx += 1

    # Copy phần còn lại của mảng con bên trái
    while l_idx < len(left):
        array[sorted_idx] = left[l_idx]
        y_positions[sorted_idx] += 1
        highlight = [sorted_idx]
        draw_array_as_squares(array, ax=ax, highlight=highlight, positions=positions, y_positions=y_positions)
        plt.pause(0.5)
        l_idx += 1
        sorted_idx += 1

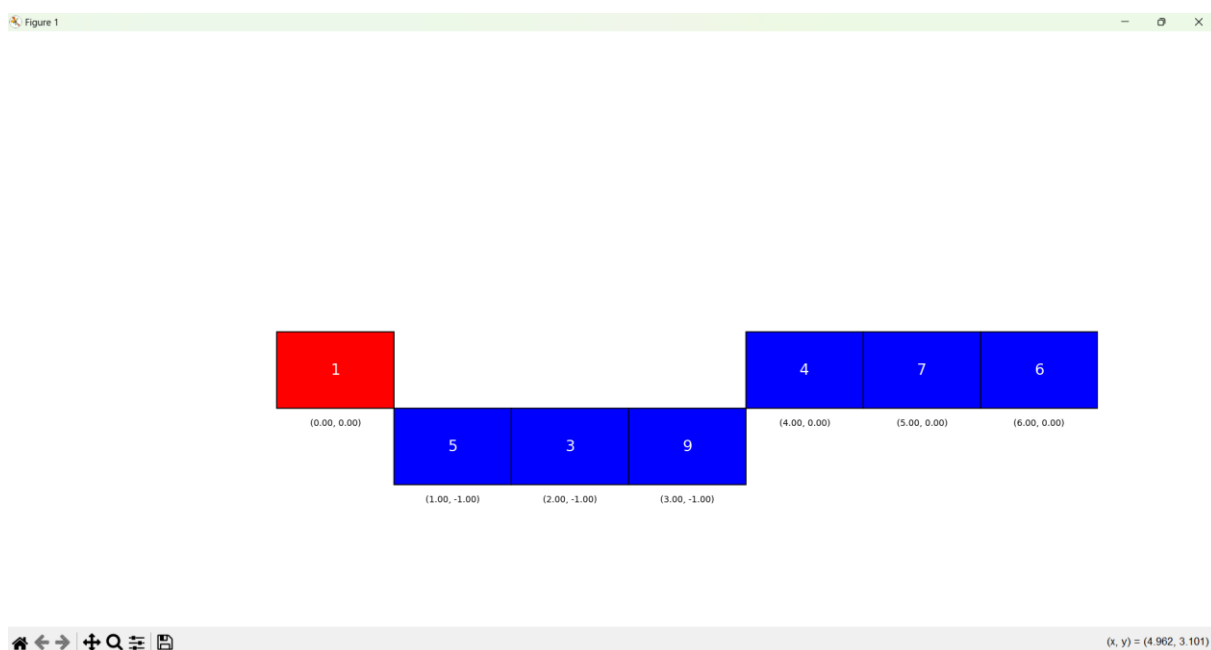
    # Copy phần còn lại của mảng con bên phải
    while r_idx < len(right):
        array[sorted_idx] = right[r_idx]
        y_positions[sorted_idx] += 1
        highlight = [sorted_idx]
        draw_array_as_squares(array, ax=ax, highlight=highlight, positions=positions, y_positions=y_positions)
        plt.pause(0.5)
        r_idx += 1
        sorted_idx += 1
```

Hình 4.5 Hàm `merge` hợp nhất hai mảng con

Hàm merge có các tham số:

- Mảng array chứa các giá trị cần hợp nhất.
- Cách chỉ số start, mid và end.
- Đối tượng Axes của Matplotlib dùng để vẽ hoạt ảnh.
- order="ascending" lựa chọn chiều sắp xếp tăng dần hoặc giảm dần, giá trị mặc định là tăng dần.
- Danh sách vị trí theo trục x (positions) và theo trục y (y_positions).

Cách thức hoạt động của hàm merge: Đầu tiên, ta chia mảng ban đầu thành hai mảng con, mảng con bên trái từ start đến mid và mảng con bên phải từ mid + 1 đến end. Thiết lập l_idx và r_idx là con trỏ chỉ số cho hai mảng con này, sorted_idx là chỉ số hiện tại trong mảng gốc nơi phần tử đã được hợp nhất sẽ được chèn vào. Ta sẽ dịch phần mảng con đang xử lý xuống bên dưới để tách biệt với mảng gốc. Sau đó, ta sử dụng vòng lặp duyệt qua các phần tử và so sánh các giá trị, nếu phần tử đó thỏa điều kiện thì sẽ được cập nhật đưa lên hợp nhất với dòng ban đầu. Nếu phần tử trong mảng left hoặc right vẫn còn thì sẽ được sao chép vào array để tiếp tục xử lý. Cứ liên tục thực hiện các thao tác cho đến khi mảng được sắp xếp.



Hình 4.6: Minh họa hợp nhất hai mảng con

2.4 Hàm vẽ cây nhị phân

Hàm draw_binary_tree sử dụng thư viện Networkx để xây dựng đồ thị có hướng (DiGraph) thể hiện một cây nhị phân có các nút và đường nối với nhau.

```

def draw_binary_tree(array, ax, n, highlight_root=None, highlight_swap=None):
    ax.clear() # Xóa nội dung trước khi vẽ lại
    G = nx.DiGraph()
    positions = {}
    edges = []

    def add_edges_and_positions(index, x, y, level_gap):
        if index >= n: # Chỉ vẽ các phần tử trong phạm vi heap
            return
        G.add_node(index, value=array[index])
        positions[index] = (x, y)
        left_child = 2 * index + 1
        right_child = 2 * index + 2
        if left_child < n:
            G.add_edge(index, left_child)
            edges.append((index, left_child))
            add_edges_and_positions(left_child, x - level_gap, y - 1, level_gap / 2)
        if right_child < n:
            G.add_edge(index, right_child)
            edges.append((index, right_child))
            add_edges_and_positions(right_child, x + level_gap, y - 1, level_gap / 2)

    add_edges_and_positions(0, 0, 0, 0.7)

    # Đảm bảo nút gốc luôn có màu đỏ và làm nổi bật các nút đổi chỗ
    node_colors = []
    for node in G.nodes():
        if highlight_root is not None and node == highlight_root:
            node_colors.append('red') # Nút gốc màu đỏ
        elif highlight_swap is not None and (node == highlight_swap[0] or node == highlight_swap[1]):
            node_colors.append('yellow') # Nút đang đổi chỗ màu vàng
        else:
            node_colors.append('skyblue')

    # Vẽ cây mà không thay đổi các đường nối
    nx.draw(G, pos=positions, ax=ax, with_labels=False, node_size=1000, node_color=node_colors, edgelist=edges)
    labels = {node: G.nodes[node]['value'] for node in G.nodes()}
    nx.draw_networkx_labels(G, pos=positions, labels=labels, ax=ax, font_size=8, font_color='black')

```

Hình 4.7: Hàm draw_binary_tree

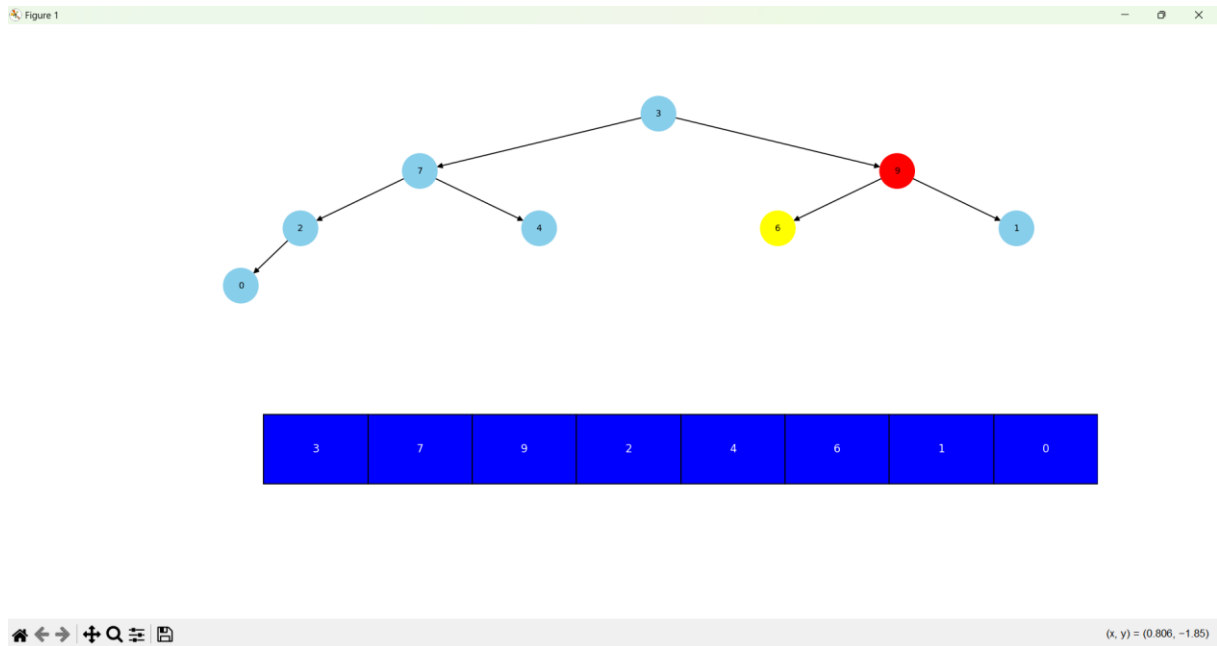
Hàm draw_binary_tree có các tham số:

- Mảng array chứa các giá trị đại diện cho cây nhị phân, mỗi giá trị là một nút trong cây.
- Đối tượng Axes từ Matplotlib, dùng để vẽ cây nhị phân.
- Số lượng phần tử n.
- Chỉ số nút gốc root (highlight_root).
- Bộ đôi chỉ số của 2 nút cần đổi chỗ (highlight_swap).

Cách thức hoạt động của hàm draw_binary_tree: Ta xây dựng một hàm con bên trong, hàm add_edges_and_positions lần lượt thêm các nút và cạnh vào cây nhị phân, mỗi nút có vị trí x và y được tính toán dựa trên mức của các nút (level).

Công thức tính chỉ số nút con:

- Nút con bên trái: $\text{left_child} = 2 * \text{index} + 1$
- Nút con bên phải: $\text{right_child} = 2 * \text{index} + 2$



Hình 4.8: Cây nhị phân vẽ bằng hàm `draw_binary_tree`

2.5 Hàm di chuyển của các nút

Hàm `move_node` thực hiện hiệu ứng di chuyển trực quan giữa hai nút trong cây nhị phân, bằng cách vẽ liên tục vẽ các nút tại vị trí hiện tại đến vị trí mới để thể hiện các mà các nút di chuyển trong cây nhị phân. Đồng thời hiển thị trạng thái của mảng tương ứng

```
def move_node(array, i, j, ax_tree, ax_array, n):
    positions = {}
    G = nx.DiGraph()
    edges = []

    # Xây dựng cây nhị phân
    def add_edges_and_positions(index, x, y, level_gap):
        if index >= n:
            return
        G.add_node(index, value=array[index])
        positions[index] = (x, y)
        left_child = 2 * index + 1
        right_child = 2 * index + 2
        if left_child < n:
            G.add_edge(index, left_child)
            edges.append((index, left_child))
            add_edges_and_positions(left_child, x - level_gap, y - 1, level_gap / 2)
        if right_child < n:
            G.add_edge(index, right_child)
            edges.append((index, right_child))
            add_edges_and_positions(right_child, x + level_gap, y - 1, level_gap / 2)
    add_edges_and_positions(0, 0, 0, 0.7)

    # Tạo hiệu ứng di chuyển giữa các nút
    steps = 20
    x1, y1 = positions[i]
    x2, y2 = positions[j]

    for step in range(steps + 1):
        alpha = step / steps
        x = (1 - alpha) * x1 + alpha * x2
        y = (1 - alpha) * y1 + alpha * y2

        positions[i] = (x, y)
        positions[j] = (x2 - alpha * (x2 - x1), y2 - alpha * (y2 - y1))

    node_colors = ['red' if node == i or node == j else 'skyblue' for node in G.nodes()]
    ax_tree.clear() # Xóa cây trước khi vẽ lại
    nx.draw(G, pos=positions, ax=ax_tree, with_labels=False, node_size=1000, node_color=node_colors, edgelist=edges)
    labels = {node: G.nodes[node]['value'] for node in G.nodes()}
    nx.draw_networkx_labels(G, pos=positions, labels=labels, ax=ax_tree, font_size=8, font_color='black')

    draw_array(array, n, ax_array)
    plt.pause(0.05) # Thời gian tạm dừng giữa mỗi bước di chuyển
```

Hình 4.9: Hàm `move_node`

Hàm `move_node` có các tham số:

- Mảng `array` chứa các phần tử nút của cây nhị phân.
- Chỉ số của 2 nút cần thực hiện di chuyển trong cây nhị phân.
- Đối tượng Axes của Matplotlib, dùng để vẽ cây nhị phân (`ax_tree`) và mảng số tương ứng (`ax_array`).
- Số lượng phần tử `n`.

Cách thức hoạt động của hàm `move_node`: Ta sử dụng hàm đệ quy `add_edges_and_positions` để thêm lần lượt các nút và cạnh vào cây nhị phân. Ta tính toán vị trí của mỗi nút trong từng bước di chuyển, vị trí hiện tại được tính bằng cách chia đều khoảng cách giữa vị trí bắt đầu (`x1, y1`) và kết thúc (`x2, y2`) theo tỉ lệ $\alpha = \text{step} / \text{steps}$, tại mỗi bước di chuyển ta sẽ phải cập nhật vị trí của nút `i` và `j`.

2.6 Hàm hiệu chỉnh Heap

Hàm `heapify` sẽ thực hiện chức năng hiệu chỉnh cây nhị phân sau mỗi lần nút gốc được đưa vào mảng sắp xếp. Sau mỗi lần nút gốc được đưa vào mảng, phần tử cuối mảng sẽ được đưa lên trên cùng làm nút gốc, lúc này hàm `heapify` sẽ hiệu chỉnh lại cây nhị phân sao cho đúng với nguyên tắc nút cha luôn lớn hơn nút con.

```
def heapify(array, n, i, ax_tree, ax_array, order="ascending"):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    # So sánh theo thứ tự tăng dần hoặc giảm dần
    if left < n and (array[left] > array[largest] if order == "ascending" else array[left] < array[largest]):
        largest = left

    if right < n and (array[right] > array[largest] if order == "ascending" else array[right] < array[largest]):
        largest = right

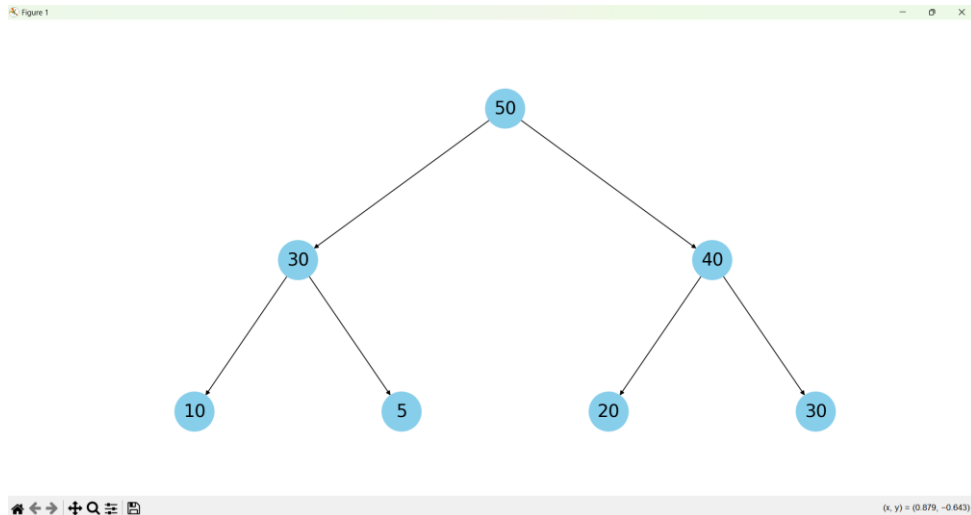
    if largest != i:
        # Hiển thị các nút đổi chỗ với hiệu ứng di chuyển
        move_node(array, i, largest, ax_tree, ax_array, n)
        array[i], array[largest] = array[largest], array[i]
        draw_binary_tree(array, ax_tree, n, highlight_root=i, highlight_swap=(i, largest))
        draw_array(array, n, ax_array)
        plt.pause(1)
    heapify(array, n, largest, ax_tree, ax_array, order)
```

Hình 4.10: Hàm hiệu chỉnh `heapify`

Hàm `heapify` có các tham số:

- Mảng đầu vào chứa các phần tử nút của cây nhị phân
- Số lượng `n` các phần tử hiện tại trong cây
- Chỉ số hiện tại của nút cần hiệu chỉnh
- `order="ascending"` lựa chọn chiều sắp xếp tăng dần hoặc giảm dần, giá trị mặc định là tăng dần.
- Đối tượng Axes của Matplotlib, dùng để vẽ cây nhị phân (`ax_tree`) và mảng số tương ứng (`ax_array`).

Cách thức hoạt động của hàm heapify: Ta xác định nút lớn nhất largest bằng cách so sánh với 2 nút con. Nếu chưa đúng nguyên tắc nút cha lớn hơn 2 nút con thì sẽ gọi hàm move_node để thực hiện hoạt ảnh di chuyển giữa hai nút i và largest trong cây. Tiếp tục gọi hàm đệ quy tại các nút con để hiệu chỉnh heap cho toàn cây nhị phân.



Hình 4.11: Minh họa cây nhị phân đã hiệu chỉnh heap

3. Module Sort

3.1 Hàm trực quan của sắp xếp chọn

Hàm selection_sort_visualize 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 các hàm swap_animation_selection và draw_array_as_squares để thể hiện các bước hóa đổi của thuật toán sắp xếp chọn.

```
def selection_sort_visualize(array, order="ascending"):
    fig, ax = plt.subplots(figsize=(10, 4))
    for i in range(len(array)):
        min_idx = i
        for j in range(i + 1, len(array)):
            draw_array_as_squares(array, highlight=[i, j], ax=ax, positions=list(range(len(array))), y_positions=[0] * len(array))
            plt.pause(0.5)
            if (order == "ascending" and array[j] < array[min_idx]) or (order == "descending" and array[j] > array[min_idx]):
                min_idx = j
        if min_idx != i:
            swap_animation_selection(array, i, min_idx, ax)
            draw_array_as_squares(array, highlight=[], ax=ax, positions=list(range(len(array))), y_positions=[0] * len(array))
    plt.show()
```

Hình 4.12: Hàm selection_sort_visualize

3.2 Hàm trực quan của sắp xếp chèn

Hàm insertion_sort_visualize 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, hàm sẽ gọi swap_animation_insertion để đổi chỗ 2 phần tử này. Quá trình so sánh và dịch chuyển này tiếp tục cho đến khi mảng được sắp xếp hoàn chỉnh. Sau khi thực hiện xong, hàm sẽ gọi draw_array_as_squares để vẽ lại mảng hoàn chỉnh.

```
def insertion_sort_visualize(array, order="ascending"):
    fig, ax = plt.subplots(figsize=(10, 4))
    positions = list(range(len(array))) # Vị trí ban đầu trên trục X
    y_positions = [0] * len(array) # Vị trí trên trục Y (mặc định là 0)

    for i in range(1, len(array)):
        key = array[i]
        j = i - 1
        if order == "ascending":
            while j >= 0 and array[j] > key:
                swap_animation_insertion(array, j + 1, j, ax)
                j -= 1
        elif order == "descending":
            while j >= 0 and array[j] < key:
                swap_animation_insertion(array, j + 1, j, ax)
                j -= 1
        array[j + 1] = key
    draw_array_as_squares(array, ax=ax, positions=positions, y_positions=y_positions)
    plt.show()
```

Hình 4.13: Hàm insertion_sort_visualize

3.3 Hàm trực quan của sắp xếp nổi bọt

Hàm bubble_sort_visualize 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 gọi hàm swap_animation_bubble để 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 draw_array_as_squares để hoàn chỉnh mảng số

```
def bubble_sort_visualize(array, order="ascending"):
    fig, ax = plt.subplots(figsize=(10, 4))
    positions = list(range(len(array))) # Vị trí ban đầu trên trục X
    y_positions = [0] * len(array) # Vị trí trên trục Y (mặc định là 0)

    # Thuật toán Bubble Sort
    for i in range(len(array) - 1):
        for j in range(len(array) - 1 - i):
            if (order == "ascending" and array[j] > array[j + 1]) or \
                (order == "descending" and array[j] < array[j + 1]):
                swap_animation_bubble(array, j, j + 1, ax)

    # Vẽ lại toàn bộ mảng sau khi sắp xếp xong
    draw_array_as_squares(array, ax=ax, positions=positions, y_positions=y_positions)
    plt.show()
```

Hình 4.14: Hàm bubble_sort_visualize

3.4 Hàm trực quan của sắp xếp trộn

Hàm merge_sort_visualize sẽ liên tục gọi đệ quy để chia đôi và sắp xếp lại mảng theo thứ tự. Sau đó, dần dần ghép từ hai mảng con nhỏ nhất cho đến khi thành một mảng hoàn chỉnh.

```
def merge_sort_visualize(array, order="ascending"):
    fig, ax = plt.subplots(figsize=(12, 8))
    positions = list(range(len(array))) # Vị trí x của các phần tử
    y_positions = [0] * len(array) # Vị trí y ban đầu

    draw_array_as_squares(array, ax=ax, positions=positions, y_positions=y_positions)
    plt.pause(0.5)

    merge_sort_recursive(array, 0, len(array) - 1, ax, positions, y_positions, order)

    draw_array_as_squares(array, ax=ax, positions=positions, y_positions=y_positions)
    plt.show()
```

Hình 4.15: Hàm merge_sort_visualize

3.5 Hàm trực quan của sắp xếp vun đống

Hàm heap_sort_visualize sẽ tạo một cây nhị phân và hiệu chỉnh heap sao cho nút cha luôn lớn hơn nút con. Tiếp theo, phần tử nút gốc sẽ được đưa vào cuối mảng. Sau đó, sẽ đưa phần tử cuối mảng lên làm nút gốc và hiệu chỉnh heap. Cứ thực hiện liên tục cho đến khi các nút trên cây nhị phân heap dần đưa vào hết trong mảng.

```
def heap_sort_visualize(array, order="ascending"):
    fig, (ax_tree, ax_array) = plt.subplots(2, 1, figsize=(12, 8))
    n = len(array)
    fig.set_size_inches(14, 10)
    # Xây dựng heap
    for i in range(n // 2 - 1, -1, -1):
        heapify(array, n, i, ax_tree, ax_array, order)

    for i in range(n - 1, 0, -1):
        array[0], array[i] = array[i], array[0]
        draw_binary_tree(array, ax_tree, i, highlight_root=0)
        draw_array(array, i, ax_array)
        plt.pause(1)

    plt.pause(1)
    draw_array(array, n, ax_array)
    heapify(array, i, 0, ax_tree, ax_array, order)

    draw_binary_tree([], ax_tree, 0)
    draw_array(array, 0, ax_array)
    plt.show()
```

Hình 4.16: Hàm heap_sort_visualize

3.6 Hàm phân hoạch cho thuật toán sắp xếp nhanh

Hàm quick_sort_partition có nhiệm vụ tách mảng số làm đôi, chọn phần tử chốt pivot là phần tử giữa mảng. Kiểm tra xem các phần tử bên trái pivot, nếu có phần tử lớn hơn chốt thì dừng lại và kiểm tra các phần tử bên phải, nếu có phần tử nhỏ hơn chốt thì dừng lại. Lúc này, left và right đã được đánh dấu tại vị trí cần đổi chỗ, gọi hàm swap_animation_quick để hoán đổi.


```
def quick_sort_partition(array, low, high, ax, positions, y_positions, order="ascending"):
    pivot_index = (low + high) // 2
    pivot = array[pivot_index]
    left = low
    right = high

    while left <= right:
        draw_array_quick(array, highlight=None, pivot=pivot_index, ax=ax,
positions=positions, y_positions=y_positions)
        plt.pause(0.05)
        # So sánh dựa trên thứ tự
        while (array[left] < pivot if order == "ascending" else array[left] > pivot):
            left += 1
        while (array[right] > pivot if order == "ascending" else array[right] < pivot):
            right -= 1
        if left <= right:
            swap_animation_quick(array, left, right, ax, pivot=pivot_index)
            left += 1
            right -= 1
    return left
```

Hình 4.17: Hàm quick_sort_partition

3.7 Hàm trực quan của sắp xếp nhanh

Hàm quick_sort_visualize sẽ gọi hàm quick_sort để thực hiện công việc phân hoạch và sắp xếp cho đến khi mảng được sắp xếp theo thứ tự.

```
def quick_sort(array, low, high, ax, positions, y_positions, order="ascending"):
    if low < high:
        pivot_index = quick_sort_partition(array, low, high, ax, positions, y_positions, order)
        quick_sort(array, low, pivot_index - 1, ax, positions, y_positions, order)
        quick_sort(array, pivot_index, high, ax, positions, y_positions, order)

    draw_array_quick(array, ax=ax, positions=positions, y_positions=y_positions)
    plt.pause(0.05)

# Hàm gọi thuật toán QuickSort và hiển thị kết quả
def quick_sort_visualize(array, order="ascending"):
    fig, ax = plt.subplots(figsize=(10, 4))
    positions = list(range(len(array))) # Vị trí ban đầu trên trục X
    y_positions = [0] * len(array) # Vị trí trên trục Y (mặc định là 0)

    quick_sort(array, 0, len(array) - 1, ax, positions, y_positions, order)
    plt.show()
```

Hình 4.18: Hàm quick_sort và quick_sort_visualize

4. Module Main

4.1 Hàm tương tác với người dùng

Hàm interaction sẽ lấy dữ liệu đầu vào từ người dùng như số lượng phần tử, giá trị các phần tử và lựa chọn thuật toán sắp xếp. Sau đó, sẽ kiểm tra các điều kiện tương ứng để hiển thị thông báo hướng dẫn cho người dùng.

```

def interaction():
    try:
        n = int(entry_n.get())
        elements = entry_elements.get().split()
        array = list(map(float, elements))

        if len(array) != n:
            result_label.config(text="Số lượng phần tử không khớp! Vui lòng thử lại (cách nhau bởi khoảng trắng).", fg="red")
            return
        elif n < 5 or n > 20:
            result_label.config(text="Số lượng phần tử trong khoảng 5 - 20", fg="red")
            return
        else:
            result_label.config(text="")

        # Lấy thuật toán được chọn
        algorithm = algorithm_var.get()
        # Lấy lựa chọn sắp xếp
        order = order_var.get()

        # Thực hiện sắp xếp và trực quan hóa
        if algorithm == "Selection Sort":
            selection_sort_visualize(array, order=order)
        elif algorithm == "Insertion Sort":
            insertion_sort_visualize(array, order=order)
        elif algorithm == "Bubble Sort":
            bubble_sort_visualize(array, order=order)
        elif algorithm == "Merge Sort":
            merge_sort_visualize(array, order=order)
        elif algorithm == "Heap Sort":
            heap_sort_visualize(array, order=order)
        elif algorithm == "Quick Sort":
            quick_sort_visualize(array, order=order)
        else:
            result_label.config(text="Vui lòng chọn một thuật toán hợp lệ!", fg="red")
    except ValueError:
        result_label.config(text="Dữ liệu không hợp lệ! Vui lòng nhập lại.", fg="red")

```

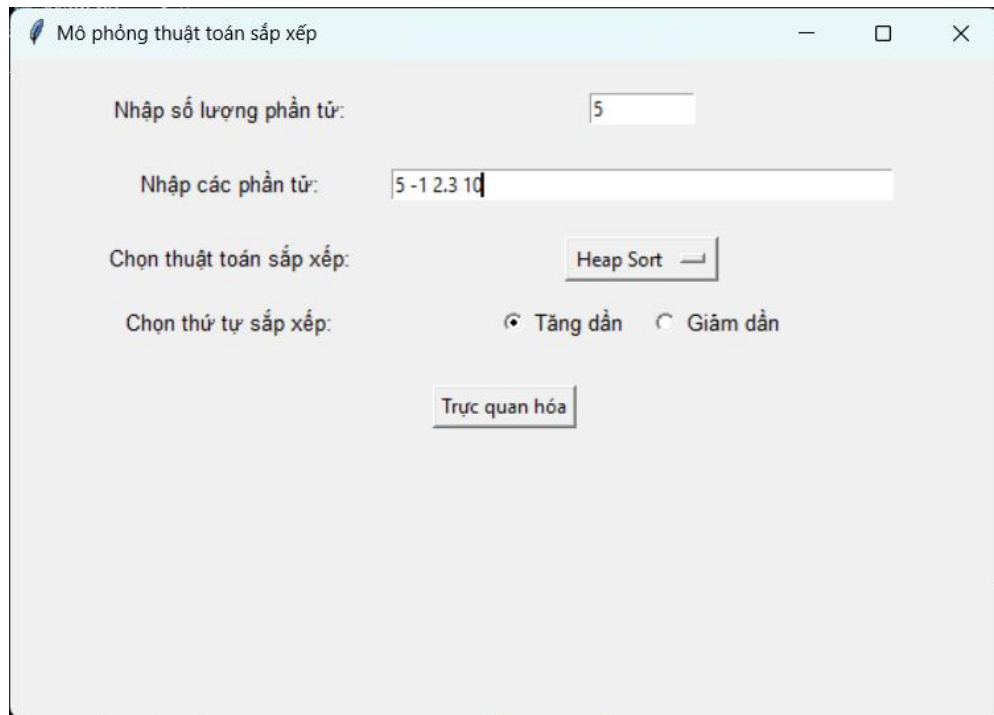
Hình 4.19: Hàm interaction

4.2 Giao diện tương tác với người dùng

Các lệnh từ thư viện Tkinter dùng để xây dựng giao diện tương tác bao gồm:

- Tạo cửa sổ chính
 - + Khởi tạo cửa sổ chính của giao diện với Tk().
 - + Đặt tiêu đề cho cửa sổ này là title("Mô phỏng thuật toán sắp xếp").
 - + Đặt kích thước của cửa sổ với chiều rộng 600 pixel và chiều cao 300pixel.
- Tạo khung nhập số lượng phần tử, giá trị các phần tử
 - + Tạo một khung (frame) để chứa các widget (Label, Entry) liên quan đến nhập liệu.
 - + Đặt khung vào cửa sổ chính, thêm khoảng cách dọc (10 pixel).
 - + Tạo label hiển thị chỉ dẫn "Nhập số lượng phần tử: ", "Nhập các phần tử: ".
- Tạo khung lựa chọn thuật toán
 - + Tạo label hiển thị chỉ dẫn "Chọn thuật toán sắp xếp:".

- + Tạo một OptionMenu để tạo menu tùy chọn các thuật toán.
- Tạo các lựa chọn
 - + Tạo label hiển thị chỉ dẫn “Chọn thứ tự sắp xếp:”
 - + Tạo ra hai lựa chọn bằng RadioButton để chọn chiều tăng dần hay giảm dần của mảng số.
- Tạo nút bấm thực hiện
 - + Tạo một nút bấm với văn bản "Trực quan hóa".
 - + Khi nhấn nút thì sẽ gọi hàm interaction để xử lý và thực hiện trực quan hóa quá trình sắp xếp.



Hình 4.20: Giao diện tương tác chính của chương trình

Chương V: KẾT LUẬN

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 hiệu ứng chuyển động, hình ảnh màu sắc dễ nhìn, khá dễ hiểu.
- Áp dụng được các thư viện Matplotlib, Networkx hay Tkinter để xây dựng chương trình.
- Có giao diện tương tác cho người dùng có thể nhập mảng tùy ý, có thể lựa chọn thuật toán mong muốn,...

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:

- Một số thuật toán có cách di chuyển và hoạt ảnh chưa đúng như mong muốn (merge sort).
- Các đối tượng hiển thị chưa được đẹp mắt
- Phân chia quản lý code còn kém, chưa thuật sự tối ưu.
- Kiến thức về các thư viện còn hạn chế, chưa áp dụng được nhiều,...

3. Kết luận

Qua quá trình thực hiện dự án, em đã vận dụng được các kiến thức đã học được trên lớp qua các môn học phần, hiểu và sử dụng được cách lập trình cơ bản của Python. Bên cạnh đó, có cơ hội tiếp xúc với các thư viện khác của Python như Matplotlib, Networkx và Tkinter. Chương trình xây dựng đã thể hiện cơ bản đúng yêu cầu đã đặt ra, thể hiện được cách hoạt động của các thuật toán tăng dần hoặc giảm dần, các đặc trưng riêng của mỗi thuật toán nhưng còn một số hạn chế.

Trong quá trình thực hiện dự án, 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 có tiếp thu, sửa đổi cho các dự án khác.

Em xin chân thành cảm ơn thầy cô.

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://viblo.asia/p/gioi-thieu-ve-matplotlib-mot-thu-vien-rat-huu-ich-cua-python-dung-de-ve-do-thi-yMnKMN6gZ7P> (viblo.asia, 21/7/2019, *Giới thiệu về Matplotlib (một thư viện rất hữu ích của Python dùng để vẽ đồ thị)*) Truy cập 28/11/2024
- [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://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