ĐẠI HỌC QUỐC GIA TP.HCM

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA KHOA HỌC & KỸ THUẬT MÁY TÍNH

ĐỀ CƯƠNG LUẬN VĂN TỐT NGHIỆP ĐẠI HỌC

TRỰC QUAN HÓA MỘT SỐ GIẢI THUẬT SẮP XẾP DỮ LIỆU

NGÀNH: KHOA HỌC MÁY TÍNH

GVHD: Th.S Trần Giang Sơn

GVPB: TS Nguyễn Hứa Phùng

---o0o---

SVTH 1: Nguyễn Đức Huy (1711510)

SVTH 2: Nguyễn Hữu Đức Thành (1613188)

TP. HỒ CHÍ MINH, 12/2020

LỜI CAM ĐOAN
---o0o---

Nhóm em xin cam đoan đề tài: **"Trực quan hóa một số giải thuật sắp xếp"** là nhóm em nỗ lực cố gắng nghiên cứu dưới sự hướng dẫn của thầy **Trần Giang Sơn**. Nhóm em xin chịu hoàn toàn trách nhiệm về tính trung thực của các nội dung trong đề tài này.

Nhóm tác giả

Nguyễn Đức Huy Nguyễn Hữu Đức Thành

LỜI CẢM ƠN
---o0o---

Nhóm tác giả **"Trực quan hóa một số giải thuật sắp xếp"** xin chân thành cảm ơn các thầy cô trong Khoa Khoa học & Kỹ thuật máy tính đã tạo điều kiện để nhóm tham gia thực hiện đề cương luận văn tốt nghiệp. Nhóm cũng xin được gửi lời cảm ơn chân thành đến thầy **Trần Giang Sơn** – Giáo viên hướng dẫn đề tài – đã định hướng và hỗ trợ nhóm tác giả. Hơn nữa, nhóm tác giả cám ơn những người bạn đã ủng hộ, động viên, khích lệ tinh thần. Nhờ những động lực từ các thầy cô và bạn bè mà nhóm mới có thể thực hiện được đề tài này.

Một lần nữa, nhóm xin chân thành cảm ơn!

Nhóm tác giả

Nguyễn Đức Huy Nguyễn Hữu Đức Thành

TÓM TẮT ĐỀ TÀI
---o0o---

Trong phạm vi đề tài “Trực quan hóa một số giải thuật sắp xếp”, ta sẽ tìm hiểu về một số giải thuật (được trình bày trong phụ lục B) và tìm hướng trực quan hóa các giải thuật này. Một số giải thuật đã được học tại trường như: Sắp xếp nổi bọt (Bubble sort), Sắp xếp chèn (Insertion sort), Sắp xếp chọn (Selection sort), Sắp xếp trộn (Merge Sort), Quick sort, Heap sort, …

Mục lục

DANH MỤC HÌNH

DANH MỤC BẢNG

Chương 1: Tổng quan về đề tài

1. Dẫn nhập và phát biểu vấn đề

Trong cuộc sống, ắt hẳn mỗi chúng ta đều đã thực hiện thao tác tìm kiếm. Ví dụ như tìm một cái áo trong tủ đồ, tìm một người bạn trong đám đông, tìm bài hát yêu thích trong danh sách nhạc của bạn,… Việc tìm kiếm này có thể mất rất nhiều thời gian. Nếu như, những thứ chúng ta đang tìm kiếm được sắp xếp theo một trật tự nào đó, việc tìm kiếm sẽ dễ dàng hơn rất nhiều. Trong khoa học máy tính cũng vậy, thao tác tìm kiếm sẽ dễ dàng hơn rất nhiều nếu dữ liệu được sắp xếp. Nhờ vậy mà chúng ta tiết kiệm được thời gian và tài nguyên hơn rất nhiều. Đó cũng là nguyên nhân mà các giải thuật sắp xếp ra đời.

Đối với các lập trình viên, giải thuật sắp xếp là một trong những giải thuật đầu tiên và chắc chắn phải học qua. Nhưng việc học các giải thuật này với các bạn mới thường rất khó khan vì tính trừu tượng và khô khan. Đó là nguồn cảm hứng cho đề tài này. Nhóm muốn tạo ra một sản phẩm có thể giúp hình dung các giải thuật sắp xếp một cách trực quan hơn, dễ nhớ, dễ tiếp thu hơn. Từ đó nâng cao năng suất dạy và học đối với nội dung này. Nhóm thực hiện rất mong chờ vào ứng dụng của sản phẩm trong tương lai.

Cụ thể, trong đề tài này, nhóm nghiên cứu một nền tảng, một công cụ hỗ trợ học tập và giảng dạy bộ môn "Cấu trúc dữ liệu và giải thuật" tại trường. Để người dùng dễ dàng tiếp cận và sử dụng, công cụ này có thể sử dụng cả trên máy tính hoặc điện thoại cá nhân. Người dùng có thể xem lại giải thuật và các ví dụ cơ bản để hiểu thêm về giải thuật đó. Công cụ cũng cung cấp các công cụ giúp trực quan hóa bằng hình ảnh các giải thuật với đầu vào do người dùng nhập vào.

2. Biện pháp giải quyết

Giải pháp mà nhóm thực hiện nhắm tới là trực quan hóa bằng biểu đồ cột và di chuyển các cột trong biểu đồ tạo thành các dòng trạng thái. Mỗi dòng trạng thái thể hiện trạng thái các phần tử trong mảng. Thông qua việc di chuyển các cột trong biểu đồ ứng với trạng thái kế tiếp, người dùng sẽ hiểu hơn về giải thuật đó.

Chương 2: Thiết kế hệ thống

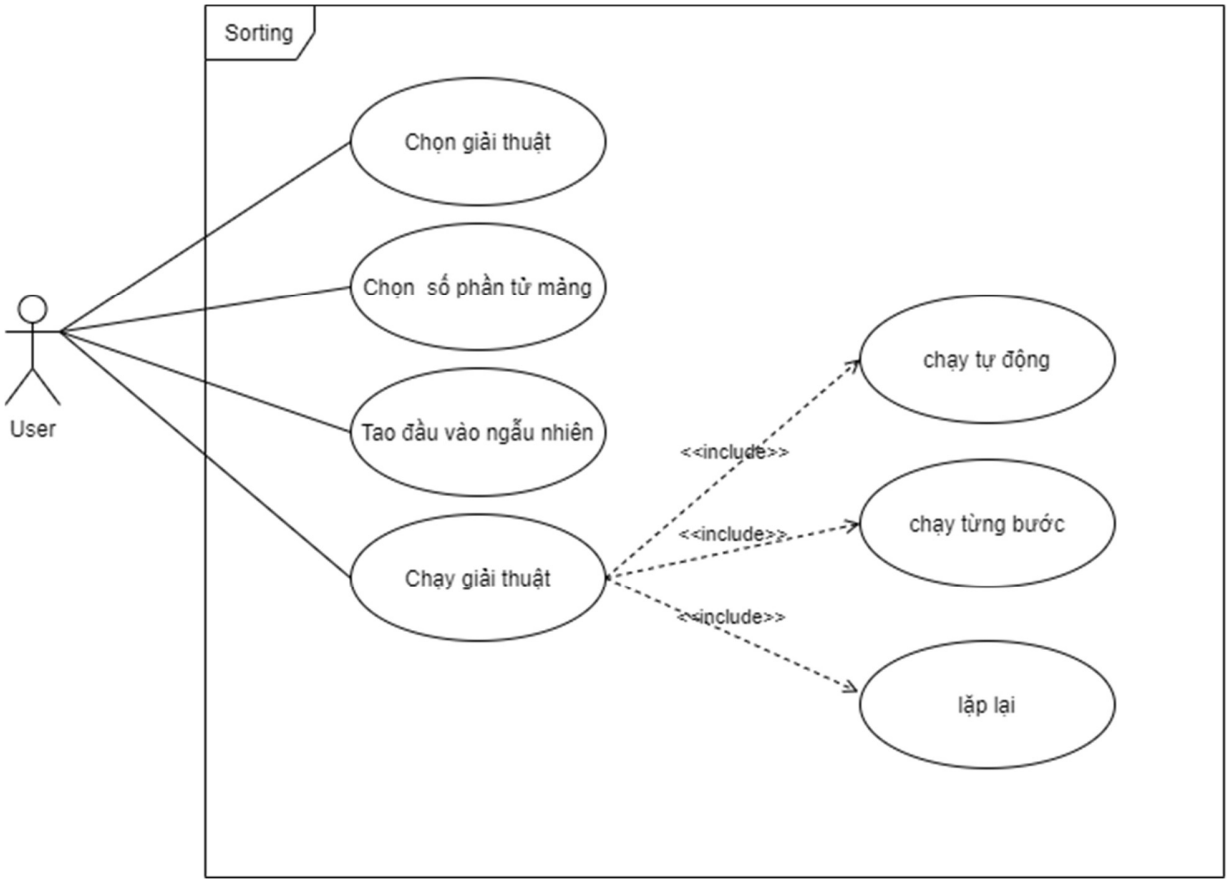
1. Công nghệ sử dụng

Nhóm hiện thực đề tài trên nền tảng web được thiết kế bằng ReactJS.

Tại sao lại là ReactJS?

- Học nhanh, dễ tiếp cận
- Cơ chế với components có thể tái sử dụng
- Render nhanh với Virtual DOM
- Nhiều công cụ phát triển, nhiều thư viện có thể sử dụng
- Có thể phát triển đề tài thành trên điện thoại với React Native

2. Use case

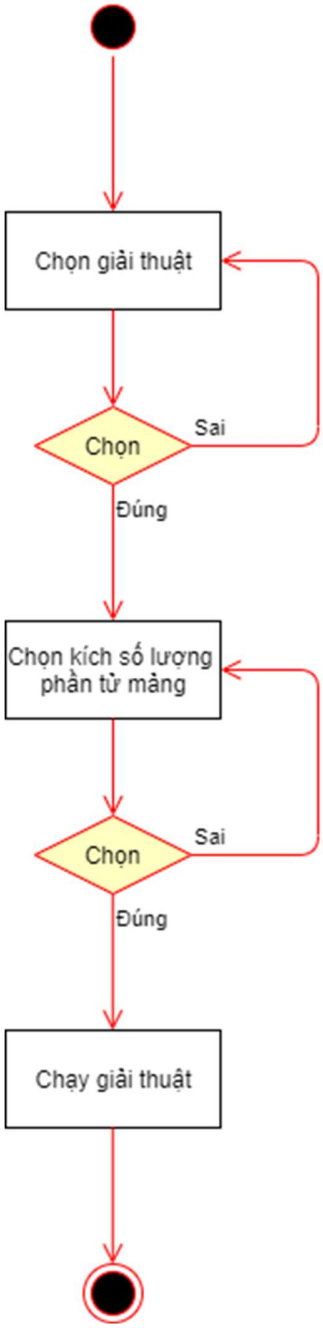


Hình 1: Usecase hệ thống

3. Use case đầy đủ và diagram

Use case	Sort visualizer		
Người tạo	Nguyễn Đức Huy	Cập nhật cuối	27/12-/2020
Đối tượng	Người dùng		
Miêu tả	Người dùng sử dụng công cụ sắp xếp		
Trigger	Người dùng truy cập trang web		
Điều kiện tiên quyết	Người dùng có thể truy cập internet		
Hậu điều kiện			
Các thao tác	1. Người dùng truy cập trang web 2. Người dùng chọn giải thuật 3. Người dùng chọn số lượng phần tử 4. Người dùng chạy giải thuật		
Ngoại lệ	1. Người dùng không chọn giải thuật → Vô hiệu hóa các chức năng khác 2. Người dùng không chọn số lượng phần tử → mặc định 10 phần tử ngẫu nhiên		

Bảng 1: Usecase đầy đủ



Hình 2: Activity diagram

4. Các yêu cầu chức năng

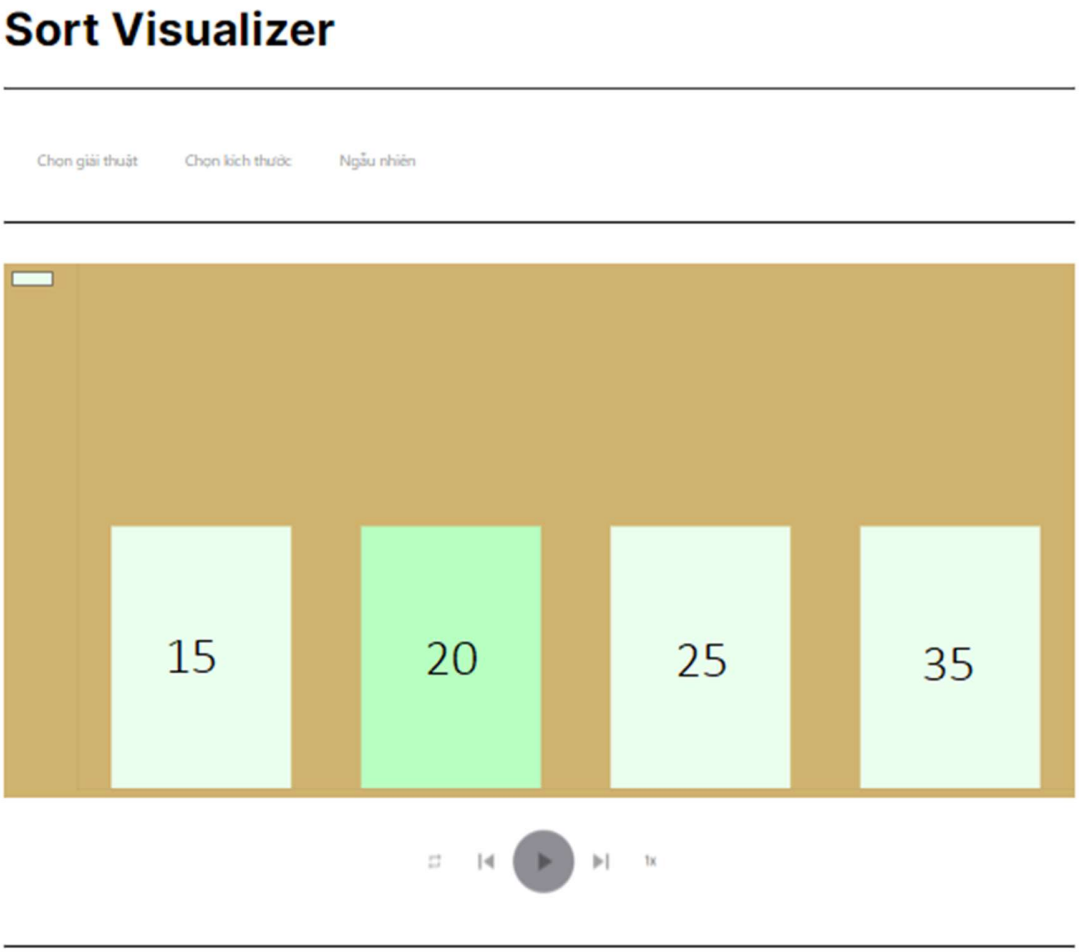
- Chọn giải thuật
- Chọn số lượng phần tử mảng đầu vào
- Tạo đầu vào ngẫu nhiên
- Xem thông tin về giải thuật (Tổng quát, độ phức tạp)

5. Các yêu cầu phi chức năng

Hoạt động tốt trên cả máy tính cá nhân và điện thoại thông minh.

Chương 3: Thiết kế giao diện

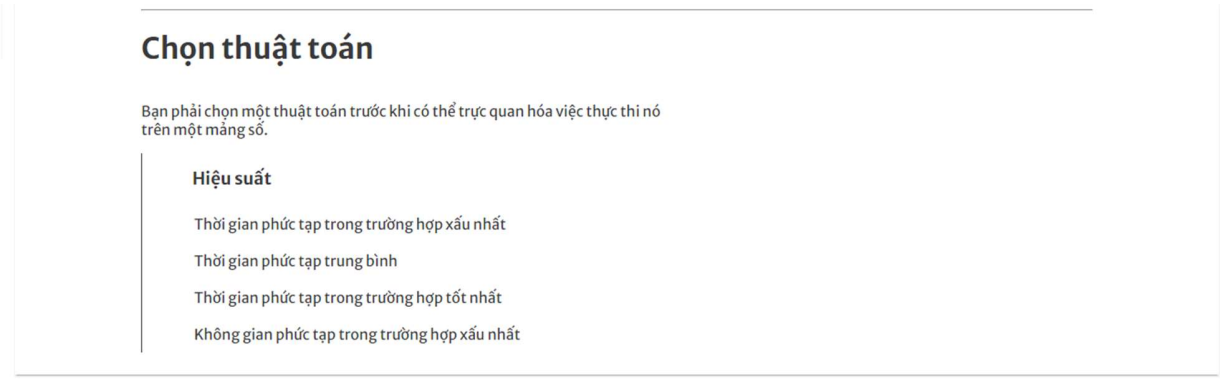
1. Giao diện biểu đồ trực quan hóa



Hình 3: Giao diện biểu đồ trựa quan

`

2. Giao diện thông tin về giải thuật

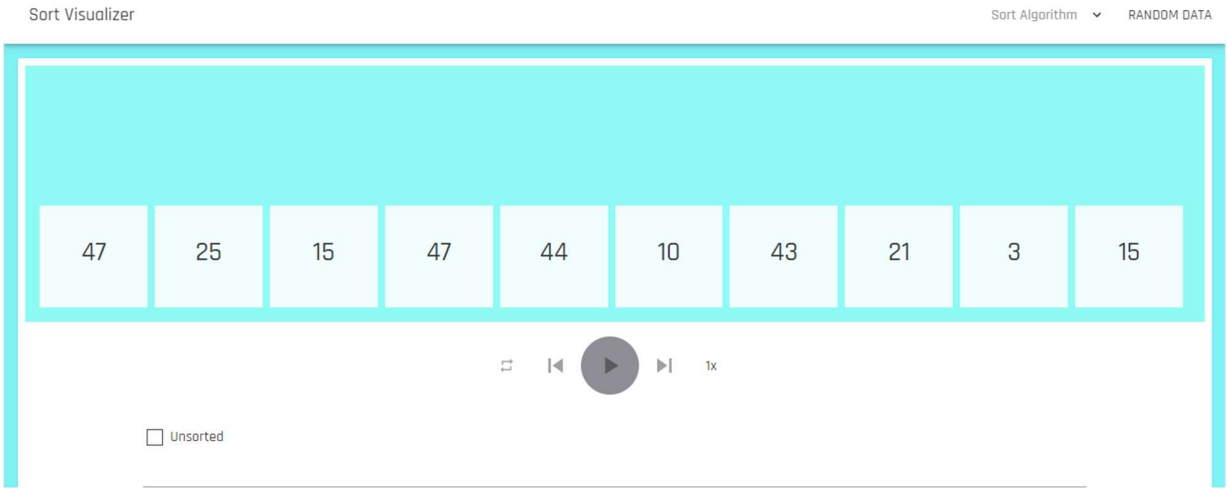


Hình 4: Giao diện thông tin về giải thuật

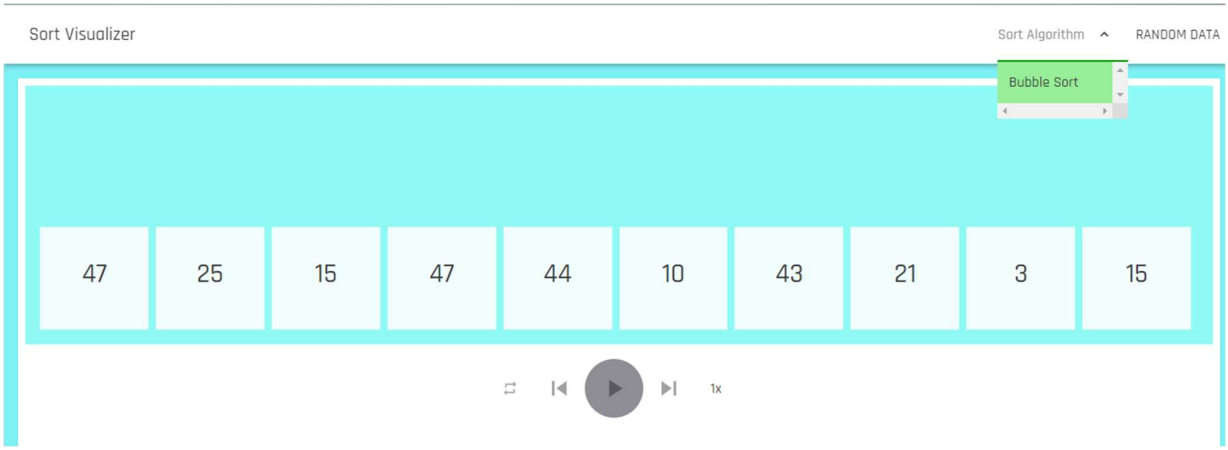
Chương 4: Hiện thực sản phẩm

1. Thiết kế giao diện và chức năng

1.1. Biểu đồ trực quan hóa dữ liệu sắp xếp



Hình 5: Giao diện biểu đồ trực quan hóa

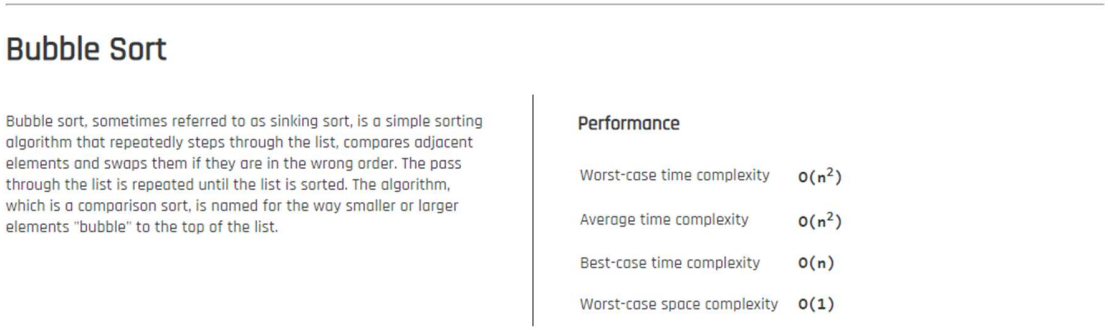


Hình 6: giao diện lựa chọn giải thuật

Mô tả:

- Khi người dùng vào trang web sẽ hiển thị như hình trên
- Để sử dụng, người dùng tiến hành chọn giải thuật tại ô Sort Algorithm sau đó, dữ liệu sẽ được tạo ngẫu nhiên.
- Bấm nút play để xem giải thuật chạy tự động, có thể tang giảm tốc độ chạy giải thuật tại ô 1x
- Người dùng cũng có thể chọn các nút chức năng bên cạnh để xem lại các bước kế cận của giải thuật đang được trực quan hóa.

1.2. Thông tin về giải thuật sắp xếp



Hình 7: Giao diện thông tin về giải thuật sắp xếp

Mô tả:

- Khi người dùng chọn một giải thuật, thì thông tin về giải thuật đó cũng được hiển thị ngay bên dưới đồ thị trực quan hóa của giải thuật đó.
- Bên trái là thông tin sơ lược về giải thuật
- Bên phải là thông tin hiệu suất của giải thuạt trong các trường hợp xấu nhất, trung bình và tốt nhất.

2. Coding

Có thể theo dõi mã nguồn trên github tại đây

Ta quan tâm đến 2 phương thức chính :

- Helper chứa các phương thức hỗ trợ
- Các giải thuật

2.1. Helper:

```
export const newTrace = (array) => {
  return [
    {
      array: [...array],
      groupA: [],
      groupB: [],
      groupC: [],
      groupD: [],
      sortedIndices: []
    }
  ];
};

export const addToTrace = (
  trace,
  array,
  sortedIndices = [],
  groupA = [],
```

```
  groupB = [],
  groupC = [],
  groupD = []
) => {
  trace.push({
    array: [...array],
    groupA: [...groupA],
    groupB: [...groupB],
    groupC: [...groupC],
    groupD: [...groupD],
    sortedIndices: [...sortedIndices]
  });
};

export const lastSorted = (trace) => {
  return trace[trace.length - 1].sortedIndices;
};

export const swap = (array, i, j) => {
  const tmp = array[i];
  array[i] = array[j];
  array[j] = tmp;
};

export const createRange = (start, end) => {
  return [...Array(end - start).keys()].map((elem) => elem + start);
};

export const createKey = (groupA, groupB, groupC, groupD) => {
  return { groupA, groupB, groupC, groupD };
};
```

3.

3.1. Giải thuật sắp xếp nổi bọt

```
import React from 'react';
import {
  swap,
  newTrace,
  addToTrace,
  lastSorted,
  createKey
} from './helpers';

const BubbleSort = (nums) => {
const trace = newTrace(nums);
```

```
  for (let i = 0; i < nums.length; i++) {
    for (let j = 0; j < nums.length - i - 1; j++) {

      addToTrace(trace, nums, lastSorted(trace), [j, j + 1]);
      if (nums[j] > nums[j + 1]) {
        swap(nums, j, j + 1);

        addToTrace(trace, nums, lastSorted(trace), [], [j, j + 1]);
      }
    }


    addToTrace(trace, nums, [
      ...lastSorted(trace),
      nums.length - 1 - i
    ]);
  }

  return trace;
};

export const BubbleSortKey = createKey('Comparing', 'Swapping');
export const BubbleSortDesc = {
  title: 'Bubble Sort',
  description: (
    <p>
      <a
        href="https://en.wikipedia.org/wiki/Bubble_sort"
        target="_blank"
        rel="noopener noreferrer"
      >
        Bubble Sort
      </a>{' '}
      is a simple sorting algorithm that repeatedly steps through the
      list, compares adjacent elements and swaps them if they are in the
      wrong order.The pass through the list is repeated until the list
      is sorted. The algorithm, which is a comparison sort, is named for
      the way smaller or larger elements "bubble" to the top of the
      list. Although the algorithm is simple, it is too slow and
      impractical for most problems
    </p>
  ),
  worstCase: (
    <span>
      O(n<sup>2</sup>)
    </span>
```

```
  ),
  avgCase: (
    <span>
      O(n<sup>2</sup>)
    </span>
  ),
  bestCase: <span>O(n)</span>,
  space: <span>O(1)</span>
};
export default BubbleSort;
```

Chương 6: Tổng kết và thu hoạch

Qua quá trình tìm hiểu và thực hiện đề tài, nhóm đã cố gắng tạo một công cụ giúp trực quan hóa các giải thuật sắp xếp. Trong quá trình thực hiên đề tài, cả hai thành viên nhóm tác giả đều được ôn tập lại các giải thuật sắp xếp, hiểu rõ hơn về các trường hợp có thể xảy ra khi ta sắp xếp dữ liệu. Cũng qua đề tài này, nhóm tác giả cũng biết thêm về lập trình web, đặc biệt là về ReactJs và triển khai dự án với NodeJs. Đây cũng là một kinh nghiệm quý báu cho hành trang tương lai của các thành viên nhóm tác giả.

Nhóm tác giả cũng tự nhận thấy các yếu tố hạn chế cần khắc phục:

- Làm việc nhóm chưa thực sự hiệu quả.
- Còn rất lúng túng khi bắt đầu với một ngôn ngữ mới.
- Tiến độ chậm, hiệu suất không được cao.

PHỤ LỤC A: Một số giải thuật sắp xếp

1. Sắp xếp nổi bọt (Bubble sort)

1.1. Định nghĩa

Sắp xếp nổi bọt là một thuật toán sắp xếp đơn giản lặp lại các bước trong danh sách, so sánh các phần tử liền kề và hoán đổi chúng nếu chúng không đúng thứ tự. Việc chuyển qua danh sách được lặp lại cho đến khi danh sách được sắp xếp. Thuật toán, là một loại so sánh , được đặt tên theo cách các phần tử nhỏ hơn hoặc lớn hơn lên đầu danh sách.

Thuật toán đơn giản này hoạt động kém trong thế giới thực và được sử dụng chủ yếu như một công cụ giáo dục.

1.2. Độ phức tạp

- Trường hợp xấu nhất và trung bình $O(n^2)$
- Trường hợp tốt nhất O(n)

1.3. Mã giả

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        swapped := false
        for i := 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap(A[i-1], A[i])
                swapped := true
            end if
        end for
    until not swapped
end procedure
```

2. Sắp xếp chèn (Insertion sort)

2.1. Định nghĩa

Sắp xếp chèn là một thuật toán sắp xếp đơn giản xây dựng mảng (hoặc danh sách) được sắp xếp cuối cùng một mục tại một thời điểm.

2.2. Độ phức tạp

- Trường hợp tốt nhất O(n).
- Trường hợp xấu nhất và trung bình $O(n^2)$.

2.3. Mã giả

```
Procedure insert (array a, int k, value) {
   int i:= k-1;
  while (i > 0 and a[i] > value) {
    a[i+1]:= a[i];
    i:= i - 1;
  }
```

```
  a[i+1]:= value;
}
```

```
Procedure InsertSort (array a, int length) {
    int k:= 2;
    while (k < length) {
        insert(a, k, a[k]);
        k:= k + 1;
    }
}
```

3. Sắp xếp chọn (Selection sort)

3.1. Định nghĩa

Sắp xếp chọn là một thuật toán sắp xếp đơn giản, dựa trên việc so sánh tại chỗ.
Chọn phần tử nhỏ nhất trong n phần tử ban đầu, đưa phần tử này về vị trí đúng là đầu tiên của dãy hiện hành. Sau đó không quan tâm đến nó nữa, xem dãy hiện hành chỉ còn n-1 phần tử của dãy ban đầu, bắt đầu từ vị trí thứ 2. Lặp lại quá trình trên cho dãy hiện hành đến khi dãy hiện hành chỉ còn một phần tử. Dãy ban đầu có n phần tử, vậy tóm tắt ý tưởng thuật toán là thực hiện n-1 lượt việc đưa phần tử nhỏ nhất trong dãy hiện hành về vị trí đúng ở đầu dãy.

3.2. Độ phức tạp $O(n^2)$

3.3. Mã giả

```
SELECTION-SORT(A)
for i = 1 to A.length - 1
        min_index = i
       for j = i + 1 to A.length
              if A[j] < A[min_index]
                     min_index = j
        key = A[i]
        A[i] = A[min_index]
       A[min_index] = key
```

4. Sắp xếp trộn (Merge sort)

4.1. Định nghĩa

Sắp xếp trộn (Merge sort) là giải thuật sắp xếp dữ liệu dựa theo chiến thuật chia để trị.

Sắp xếp trộn hoạt động như sau:

- Chia danh sách chưa được sắp xếp thành *n* danh sách con, mỗi danh sách chứa một phần tử (danh sách một phần tử được coi là đã sắp xếp).
- Liên tục hợp nhất các danh sách con để tạo ra các danh sách con được sắp xếp mới cho đến khi chỉ còn lại một danh sách con. Đây sẽ là danh sách được sắp xếp.

4.2. Độ phức tạp O(n log n)

4.3. Mã giả

```
function merge(left, right) is
    var result := empty list

    while left is not empty and right is not empty do
        if first(left) ≤ first(right) then
            append first(left) to result
            left := rest(left)
        else
            append first(right) to result
            right := rest(right)

    // Either left or right may have elements left; consume them.
    // (Only one of the following loops will actually be entered.)
    while left is not empty do
        append first(left) to result
        left := rest(left)
    while right is not empty do
        append first(right) to result
        right := rest(right)
    return result

function merge_sort(list m) is
    // Base case. A list of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide the list into equal-sized sublists
    // consisting of the first half and second half of the list.
    // This assumes lists start at index 0.
    var left := empty list
    var right := empty list
    for each x with index i in m do
        if i < (length of m)/2 then
            add x to left
        else
            add x to right

    // Recursively sort both sublists.
    left := merge_sort(left)
    right := merge_sort(right)

    // Then merge the now-sorted sublists.
```

```
    return merge(left, right)
```

5. Quick sort

5.1. Định nghĩa:

Quick sort là giải thuật sắp xếp dữ liệu dựa theo chiến thuật chia để trị.

Quick sort chia mảng thành hai danh sách bằng cách so sánh từng phần tử của danh sách với một phần tử được chọn được gọi là phần tử chốt. Những phần tử nhỏ hơn hoặc bằng phần tử chốt được đưa về phía trước và nằm trong danh sách con thứ nhất, các phần tử lớn hơn chốt được đưa về phía sau và thuộc danh sách đứng sau. Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1.

5.2. Độ phức tạp

- Trường hợp xấu nhất $O(n^2)$.
- Trường hợp trung bình và tốt nhất O(n log n).

5.3. Mã giả

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p - 1)
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo
    for j := lo to hi do
        if A[j] < pivot then
            swap A[i] with A[j]
            i := i + 1
    swap A[i] with A[hi]
    return i
```

TÀI LIỆU THAM KHẢO

[1] *https://digitalcommons.ric.edu/cgi/viewcontent.cgi?article=1129&context=honors_projects*

[2] *https://en.wikipedia.org/wiki/Sorting_algorithm*

[3] *https://reactjs.org/docs/getting-started.html*

[4] *https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html*