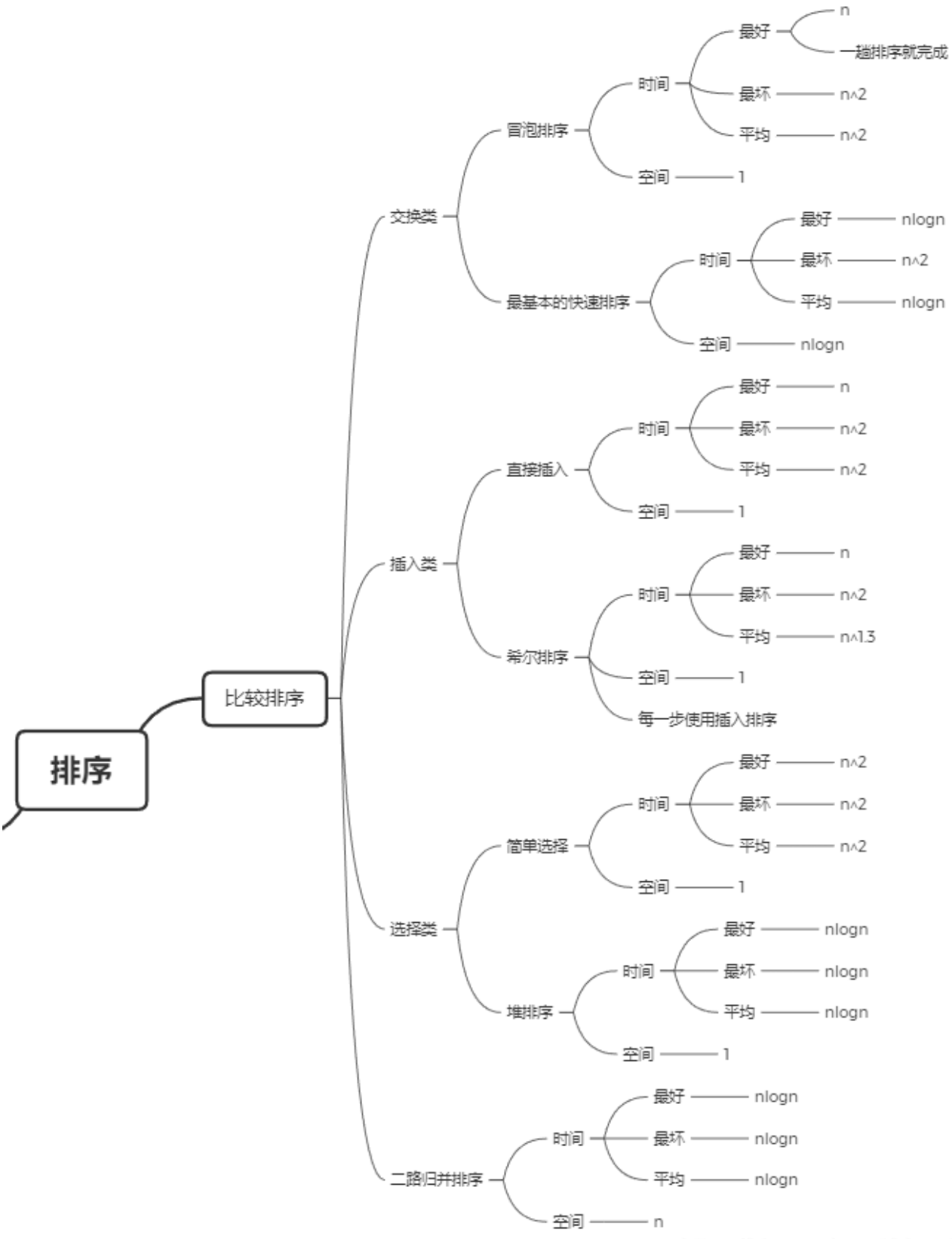
汇总

一共十种排序



所有排序方法可以在这个repo里面找到，https://github.com/hhmy27/DataStruct

冒泡

```
void bubbleSort(int *A, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (A[j] > A[j + 1]) swap(A[j], A[j + 1]);
        }
    }
}
```

快排

最基本的快排版本

```
void quickSort__(int *A, int left, int right) {
    if (left >= right)
        return;
    int val = A[left];
    int i = left, j = right;
    while (i < j) {
        for (; i < j && A[j] >= val; j--);
        A[i] = A[j];
        for (; i < j && A[i] <= val; i++);
        A[j] = A[i];
    }
    A[i] = val;
    quickSort__(A, left, i - 1);
    quickSort__(A, i + 1, right);
}

// 函数入口
void quicksort(int *A, int n) {
    quickSort__(A, 0, n - 1);
}
```

直接插入

```
void insertSort(int *A, int n) {
    for (int i = 1; i < n; i++) {
        int v = A[i];
        int j = i - 1;
        for (; j >= 0 && A[j] >= v; j--)
            A[j + 1] = A[j];
        A[j + 1] = v;
    }
}
```

希尔

```
// we choose initial m equal floor n/2
// in the inside sort, we use insert sort method
```

```c
void shellSort(int *A, int n) {
    int m = n / 2;
    while (m) {
        // sort inside array
        for (int i = 0; i < n; i += m) {
            // use insert sort
            for (int j = i + m; j < n; j += m) {
                int v = A[j];
                int k = j - m;
                for (; k >= 0 && A[k] >= v; k -= m)
                    A[k + m] = A[k];
                A[k + m] = v;
            }
        }
        // keep sort inside array until m = 0
        m -= 1;
    }
}
```

简单选择

```c
void selectSort(int *A, int n) {
    for (int i = 0; i < n; i++) {
        int m = A[i];
        int ind = i;
        for (int j = i + 1; j < n; j++) {
            if (A[j] < m) {
                m = A[j];
                ind = j;
            }
        }
        swap(A[i], A[ind]);
    }
}
```

堆排序

```c
void heapSort(int *A, int n) {
    // shallow copy, O(n)
    MaxHeap heap = createMaxHeap(A, n);

    for (int i = 0; i < n; i++) {
        // max number swap to tail
        int temp = heap->array[0];
        heap->array[0] = heap->array[heap->size - 1];
        heap->array[heap->size - 1] = temp;
        heap->size -= 1;
        // O(logn)
        maxHeapify(heap, 0);
    }
}
```

其中MaxHeap的代码如下

```c
typedef struct max_heap {
    int size;
    int *array;
} max_heap, *MaxHeap;

// construct max heap
void maxHeapify(MaxHeap heap, int ind) {
    int min_ind = ind;
    int left = 2 * ind + 1;
    int right = 2 * ind + 2;

    int *arr = heap->array;
    int size = heap->size;

    // select minimal number in root, left, right
    if (left < size && arr[left] > arr[min_ind])
        min_ind = left;
    if (right < size && arr[right] > arr[min_ind])
        min_ind = right;

    if (min_ind != ind) {
        swap(arr[min_ind], arr[ind]);
        maxHeapify(heap, min_ind);
    }
}

MaxHeap createMaxHeap(int *A, int n) {
    MaxHeap heap = (MaxHeap) malloc(sizeof(max_heap));
    heap->array = A;
    heap->size = n;

    for (int i = (n - 2) / 2; i >= 0; i--) {
        maxHeapify(heap, i);
    }

    return heap;
}

MaxHeap createMaxHeap_deep(int *A, int n) {
    MaxHeap heap = (MaxHeap) malloc(sizeof(max_heap));
    heap->array = (int *) malloc(sizeof(int) * n);
    heap->size = n;

    for (int i = 0; i < n; i++) {
        heap->array[i] = A[i];
    }

    for (int i = (n - 1) / 2; i >= 0; i--) {
        maxHeapify(heap, i);
    }

    return heap;
}

int popMaxHeapRoot(MaxHeap heap) {
    int *arr = heap->array;
    // get root number
    int n = arr[0];
    swap(arr[0], arr[heap->size - 1]);
    heap->size -= 1;
    maxHeapify(heap, 0);
```

```
    return n;
}
```

2路归并排序

```
void merge(int *A, int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1];
    int R[n2];

    for (i = 0; i < n1; i++)
        L[i] = A[left + i];
    for (j = 0; j < n2; j++) {
        R[j] = A[mid + j + 1];
    }

    i = j = 0;
    k = left;
    // compare L[i] and R[j] than assign smaller number to A[k]
    // add 1 to relevant index to iterate rest of number
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            A[k++] = L[i++];
        } else {
            A[k++] = R[j++];
        }
    }
    while (i < n1)A[k++] = L[i++];
    while (j < n2)A[k++] = R[j++];

}

void mergeSort_2__(int *A, int left, int right) {
    if (left >= right)
        return;
    int mid = (left + right) / 2;
    mergeSort_2__(A, left, mid);
    mergeSort_2__(A, mid + 1, right);
    merge(A, left, mid, right);
}


// 函数入口
void mergeSort_2(int *A, int n) {
    mergeSort_2__(A, 0, n - 1);
}
```

计数排序

```
void countingSort(int *A, int n) {
    if (n == 0)
        return;
    // use max - min create array can save space
    int max = A[0];
    int min = A[0];
    for (int i = 0; i < n; i++) {
        if (A[i] > max)
            max = A[i];
        if (A[i] < min)
            min = A[i];
    }
    int C[max - min + 1];
    memset(C, 0, sizeof(int) * (max - min + 1));

    for (int i = 0; i < n; i++) {
        C[A[i] - min]++;
    }

    int ind = 0;
    for (int i = 0; i < len(C); i++) {
        while (C[i]--) {
            A[ind++] = i + min;
        }
    }
}
```

桶排序

桶排序和基数排序中用到了LinkList，这是我自己实现的一个带头结点的单链表，下面是链接

https://github.com/hhmy27/DataStruct/blob/main/List/LinkList/SingleLinkList/SingleLinkList.h

```
void bucketSort(int *A, int n) {
    if (n == 0)
        return;
    const int BUCKET_SIZE = 5;

    // use max - min create array can save space
    int max = A[0];
    int min = A[0];
    for (int i = 0; i < n; i++) {
        if (A[i] > max)
            max = A[i];
        if (A[i] < min)
            min = A[i];
    }

    int d = max - min + 1;
    LinkList buckets[d];
    for (int i = 0; i < d; i++) {
        buckets[i] = createEmptyLinkList();
    }

    for (int i = 0; i < n; i++) {
        // map function: f(A[i]) = ((A[i] - min) / BUCKET_SIZE)
```

```
        insertNodeHead(buckets[(A[i] - min) / BUCKET_SIZE], A[i]);
    }

    // sort bucket
    for (int i = 0; i < d; i++) {
        sortLinkList(buckets[i]);
    }

    int ind = 0;
    for (int i = 0; i < d; i++) {
        Array a = (Array) transformToArray(buckets[i]);
        int *arr = a->arr;
        for (int j = 0; j < a->size; j++) {
            A[ind++] = arr[j];
        }
    }
}
```

基数排序

```
int getNumInPos(int num, int pos) {
    while (--pos) {
        num /= 10;
    }
    return num % 10;
}

// n is length , m is max length of number
void radixSort(int *A, int n, int m) {
    const int BASE = 10;
    // 10 base
    LinkList container[10];
    for (int i = 0; i < BASE; i++) {
        container[i] = createEmptyLinkList();
    }
    // process all position
    for (int i = 1; i <= m; i++) {
        // process all number in given position
        for (int j = 0; j < n; j++) {
            // get current number
            int tmp = A[j];
            // insert to corresponding contain
            insertNodeTail(container[getNumInPos(tmp, i)], tmp);
        }
        // gather number in contain
        int ind = 0;
        for (int k = 0; k < BASE; k++) {
            while (!isEmptyLinkList(container[k]))
                A[ind++] = popFrontVal(container[k]);
        }
    }

}
```