Phân Tích Thuật Toán
Trường Đại Học Khoa Học Tự Nhiên
Nguyễn Thanh Bình

Report lab 4

Nguyễn Quốc Bảo - 18110053

Bài 1. Cho mảng A đã được sắp xếp có N phần tử. Tìm một phần tử x trong A
Ta sử dùng hàm **binary-search** để tìm phần tử x trong mảng A đã sắp xếp.

```python
# Binary search algorithm implementation
def binary_search(arr,left,right,x):
    if right >= left:
        mid = left + (right - left)//2

        if (arr[mid] == x):
            return mid

        if (arr[mid] > x):
            return binary_search(arr,left,mid-1,x)

        return binary_search(arr,mid+1,right,x)

    return -1
```

Vì mỗi lần lặp đều dựa trên mid nên ta có công thức cho hàm **binary-search** là $T(n) = T(n/2) + c$.
Từ đó, ta có thể nói độ phức tạp của hàm **binary-search** là $O(logn)$.

Ta có kết quả sau khi chạy:

```
---------------------------------- Exercise 1 ----------------------------------

>> Input x :7
Input array A
>> Enter number of elements : 10

>> Enter the numbers : 3 6 9 4 1 0 7 1 10 12
Array sorted
 [0, 1, 1, 3, 4, 6, 7, 9, 10, 12]
found element 7 at 6
```

Để thử nhiều trường hợp hơn ta có bộ dữ liệu để kiểm tra.
- Cho N = 10, 20, 30,..., 100
- Tạo ngẫu nhiên A đã sắp xếp và phần tử x.

Ta có đầy đủ kết quả cho phần kiểm tra này nằm ở tệp test-ex1.txt

Bài 2. Cho tập hợp S có N phần tử khác nhau $S(i) \in \{1, 2, ..., 1000\}$. Viết chương trình tìm phần tử nhỏ nhất thứ k trong tập S ($1 \leq k \leq N$).
Để giải quyết bài toán ta thực hiện các bước sau:

1. Tạo tập S thỏa với điều kiện bài toán
2. Tìm phần tử bé nhất thứ k trong tập S.
   - Sắp xếp tập S theo thứ tự tăng dần, ở bài này ta có thể sử dụng nhiều cách sắp xếp, ở đây ta chọn heapSort để thực hiện.

Đầu tiên ta xác định cho tập S là cây nhị phân bằng hàm **heapify** với nút gốc là số lớn nhất và các nút con.

```python
def heapify(arr, n, i):
    global counter_assign
    global counter_compare

    # Find largest among root, left child and right child
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    counter_assign +=3

    if l < n and arr[i] < arr[l]:
        largest = l
        counter_assign +=1

    if r < n and arr[largest] < arr[r]:
        largest = r
        counter_assign +=1

    # If root is not largest, swap with largest and continue heapifying
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        counter_assign +=1
        heapify(arr, n, largest)

    counter_compare += 3
```

Sau đó khi hàm được áp dụng cho tất cả các phần tử bao gồm cả phần tử gốc.
Ta bắt đầu bằng cách xếp đống các cây nhỏ nhất thấp nhất và dần dần di chuyển lên cho đến khi chúng ta đến phần tử gốc.
Và chiều cao của một cây nhị phân hoàn chỉnh chứa n phần tử là log n

```python
def heapSort(arr):
    global counter_assign
    global counter_compare

    n = len(arr)

    # Build max heap
    for i in range(n//2, -1, -1):
        counter_compare +=1
        heapify(arr, n, i)

    for i in range(n-1, 0, -1):
        # Swap
        arr[i], arr[0] = arr[0], arr[i]
        counter_assign +=1
        counter_compare +=1
        # Heapify root element
        heapify(arr, i, 0)
```

Ta thực hiện xây dựng S là cây nhị phân cho $n/2$ phần tử nên độ phức tạp trong trường hợp xấu nhất của bước này là $n/2 * logn \sim nlogn$.
Trong bước tiếp theo ta hoán đổi nút gốc lớn nhất và phần tử cuối. Đối với mỗi phần tử, điều này lại làm mất thời gian xấu nhất là log n vì chúng ta có thể phải đưa phần tử đó đi suốt từ gốc đến lá. Và ta lặp lại n lần nên bước sắp xếp này nên số lần lặp của bước này là nlog n.
Vì 2 bước trên thực hiện lần lượt nên độ phức tạp của thuật toán heapSort là nlog n

```python
# Function find k'th smallest element in a given array
def kth_smallest(arr, k):
    global counter_assign
    global counter_compare

    # Sort the given array
    heapSort(arr)

    # Return k'th element in the
    # sorted array
    return arr[k-1]
```

Sau khi đã sắp xếp xong ta chỉ cần trả về phần tử thứ k trong tập S đã sắp xếp.
Ta có kết qua sau khi chạy:

```
---------------------------------- Exercise 2 ----------------------------------

--------------------------------------------------------------------------------
Array sorted
 [45, 139, 161, 170, 210, 220, 490, 607, 641, 861]
N =  10
Element 210 is 5th smallest
count_compare: 105 and count_assign: 134
--------------------------------------------------------------------------------
Array sorted
 [1, 37, 66, 153, 167, 171, 253, 394, 547, 584, 628, 648, 670, 759, 780, 801, 871, 875, 916, 992]
N =  20
Element 167 is 5th smallest
count_compare: 297 and count_assign: 425
```

kết quả đầy đủ nằm trong tệp result-ex2.txt