Advance Binary search

Tags	Algorithm, Algorithm-Design, Binary-Search, Divide&Conquer, MyNotes, Problem-Solving approach, Searching
Companies	Amazon, Google, INFOSYS, Microsoft
Created time	@November 1, 2022 6:21 PM
Last edited time	@November 3, 2022 9:16 AM
Status	Completed
URL	https://www.interviewbit.com/problems/allocate-books/
URL 1	

Its an concept to find best answer in search space using monotonic predicate function with binary search.

> 💡 search space - its a range of values that might be the element we are looking for.

> 💡 monotonic predicate function -predicate function are function which gives boolean output. monotonic predicate function are subset of predicate function with an extra feature which is it always gives set of output in a monotonic manner (increasing/decreasing). because its boolean and monotonic nature its output set relative to search space will always look like either […,T,T,T,T,T,T,F,F,F,F,...] or […,F,F,F,F,F,T,T,T,…]. its like

> collection that returns false before a point/input and true and vice versa.

Why do we need this? (Use-case) and how this works?

We use binary search mostly to find solution in a monotonic input list. but input doesn't always have to monotonic input. we can apply binary search on answers. suppose you are given a problem which has range of solution and we have to the minimum or maximum element. all we have to do is find the search space range which doesn't always have be very precious since we are using logarithmic solution it will add one or two more iterations. and design a predicate function corresponding to the question. after that all we have to do is fit them in binary search if the task is to find the maximum whenever the predicate function return true keep it in answer variable and binary search in right partition [mid+1,up] else search in left partition [0,mid-1].

Some famous example questions

▼ **Book Allocation Problem 🔗**

▼ Description

Given an array of integers **A** of size **N** and an integer **B**. College library has **N** bags,the I **th** book has **A[i]** number of pages. You have to allocate books to **B** number of students so that maximum number of pages allotted to a student is minimum.

▼ Solution

search space - the minimum number of pages that can be allotted is the no of page of book with minimum pages and the maximum page will be the sum of all book pages.

predicate function - function will take array, no of student, and the test answer. in this function we have to iterate thru the array while keeping count of student and no of book pages in each iteration firstly check after adding ith book to student no 'y', will it cross the 'test_answer' if not allocate that book to Yth student. if it is crossing the 'test_answer' limit increment student variable(means we have allocated the no of books to Yth student) reset the no of books to 0, after all of that check if we have crossed the number of student if we have just return false (because it means the test_answer is too low for the no of books to be allocated) and if we have successfully

completed all the iterations return true because we have allocated all the books.

binary search - set lo as start of search space itc min of the input array then set hi as end of search space itc sum of the input array. calculate mid check predicate if true keep it in a variable answer that we can return after search after that set hi as mid-1 because after it all the cases in search space will give right answer but we have to find the minimum. if false set lo as mid+1 because if mid cannot be the answer no number lesser than mid will have chance to give the answer after the search return the answer variable(note declare it before loop so it doesnt go out of scope)

▼ code

```
bool possible(vector<int> &A,int students,int mid)
{
    int studentCount{1};
    int pageCount{0};
    for(int i = 0;i < A.size();i++)
    {
        if(pageCount+A[i] <= mid)
        {
            pageCount+=A[i];
        }
        else
        {
            studentCount++;
            pageCount = A[i];
            if(studentCount > students || pageCount > mid) return 0;
        }
    }
    return true;
}


int Solution::books(vector<int> &A, int B) {
    if(B>A.size()) return -1;
    int start = 0;
    int end = 0;
    int answer = -1;
    for(auto&&i:A)end+=i;
    while(start<=end)
    {
        int mid = start + (end-start)/2;
        if(possible(A,B,mid))
        {
            answer = mid;
            end = mid-1;
        }
        else start = mid + 1;
    }
```

```
    return answer;
}
```

▼ **Painter partition Problem** 🔗

▼ Description

Given 2 integers **A** and **B** and an array of integers **C** of size **N**.

Element **C[i]** represents length of **ith** board.

You have to paint all **N** boards **[C0, C1, C2, C3 … CN-1]**. There are **A** painters available and each of them takes **B** units of time to paint `1 unit` of board.

Calculate and return minimum time required to paint all boards under the constraints that **any painter will only paint contiguous sections of board.**

- 2 painters cannot share a board to paint. That is to say, a board cannot be painted partially by one painter, and partially by another.
- A painter will only paint contiguous boards. Which means a configuration where painter 1 paints board 1 and 3 but not 2 is invalid.

▼ Solution

Similar as book allocation

▼ Code

```
bool check(vector<int> &boards,int cap,int partation)
{
    int sum{0};
    int x{1};
    for (size_t i = 0; i < boards.size(); i++)
    {
        if(sum+boards[i] <= partation)
        {
            sum+=boards[i];
        }
        else
        {
            x++;sum = boards[i];
            if(x > cap || boards[i] > partation) return false;
        }
    }
    return true;
}
int findLargestMinDistance(vector<int> &boards, int k)
{
    if(boards.size() == k) return *max_element(boards.begin(),boards.end());
    int up{0};
```

```
    int total{0};
    for (size_t i = 0; i < boards.size(); i++)
    {
        total+=boards[i];
    }

    int down{total};
    int answer{-1};
    while (up<=down)
    {
        int mid = (up+down)/2;
        if(check(boards,k,mid))
        {
            down = mid-1;
            answer = mid;
        }
        else
        {
            up = mid+1;
        }
    }
    return answer;
}
```

▼ **Aggressive Cows Problem 🔗**

▼ Description

Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000). His C (2 <= C <= N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

▼ Solution

Search Space - 0 to max of array

Predicate function - given(Array,cowNum,TestAnswer) keep a variable cow placed set it to one indicating we have already placed a cow in arr[0]th stall and keep a current position set it to arr[0] because we have placed a cow already there now iterate thru Array check if adding arr[i] distance to current position sustain TestAnswer distance if it does increment cow-placed variable and set current position to arr[i] so in next iteration it will check distance against this position if now just move on to next iteration but before ending loop always check if cow-placed == total cow we have if yes return true because we have placed all cow. and if we have completed all the

iteration return false because we have checked all the position with that minimum distance(test-answer) and we have not successfully placed all cows

Binary search - every time predicate function return true set lo to mid+1 else because if we can place cows with x distance means we can place cow lo to x and we have to find the maximum distance else set hi to mid-1.

▼ code

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

bool isPossible(vector<int>& pos,int size,int cows,int PS){
  int cc = 1;//cow count
  int cp = pos[0];//current position
  for(int i = 0;i < size;i++){
    if(pos[i]-cp >= PS){
      cp = pos[i];
      cc++;
    }
    if(cc==cows) return true;
  }
  return false;
}

int AGGRCOW(vector<int>& pos,int size,int cows){
  sort(pos.begin(),pos.end());
  int lo = 0;
  int hi = pos[size-1];
  int ans = -1;
  while(lo<=hi){
    int mid = lo + (hi-lo)/2;
    if(isPossible(pos,size,cows,mid))
    {
      ans = mid;
            lo = mid+1;
    }
    else
      hi = mid-1;
  }
  return ans;
}

int main() {
  int q;cin>>q;
  while(q--){
    int size; cin>>size;
    int cows; cin>>cows;
    vector<int> pos(size,0);
    for(auto&&i:pos) cin>>i;
    cout << AGGRCOW(pos,size,cows) << endl;
```

```
  }
  return 0;
}
```

▼ Square root(integer part)

search space - 1 to Num

predicate function - check if Mid from bs when squaring it is smaller than or equal to the Num and return true if it is;

binary search - if mid when squaring is equal to Num return mid else if it is bigger then set hi to mid-1 else set lo to mid+1