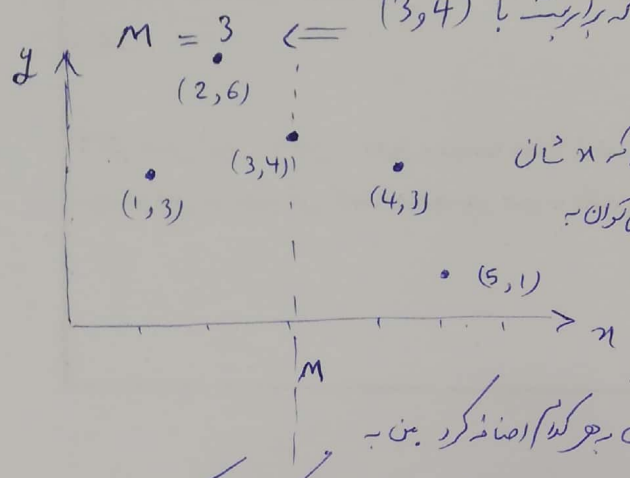
1403 DA - HW1 علیرضا کریمی ۸۱۰۱۰۱۴۹۲

۱) ابتدا یکبار برحسب x و بار دیگر برحسب y، مرتب می‌کنیم ⇐

$P_x = \{(1,3), (2,6), (3,4), (4,3), (5,1)\}$

$P_y = \{(5,1), (1,3), (4,3), (3,4), (2,6)\}$

حال نزدیک‌ترین زوج برابر خواهد بود با راست‌ترین زوج آرایه P_x که برابر است با $(3,4)$ و $(4,3)$ به صورت عمودی



تمام نقاطی که x آن‌ها کوچک‌تر از M است در سمت چپ و آن‌هایی که x شان بزرگ‌تر از M است در سمت راست. آن‌هایی که $x=M$، می‌توان هر طرف اضافه کرد ولی اولویت با طرفی است که عضو کمتری دارد که تعداد اعضای چپ و راست خط، بالانس باشند (اینجا چون هر دو طرف یک تعداد راسِ یکسان دارند، می‌توان به هرکدام اضافه کرد. من به سمت سمت چپ اضافه می‌کنم) حال دوباره باید مسئله را برای هر دو بخش به صورت بازگشتی حل کرد. حال داریم:

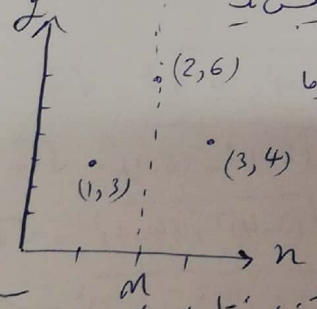
سمت راست: چون تنها دو نقطه داریم، فاصله min می‌شود فاصله میان آن دو ⇐ $\sqrt{(5-4)^2+(1-3)^2} = \sqrt{5}$

سمت چپ: چون بیشتر از دو نقطه داریم، روند تقسیم را ادامه می‌دهیم. نقاط چپ عبارت‌اند از

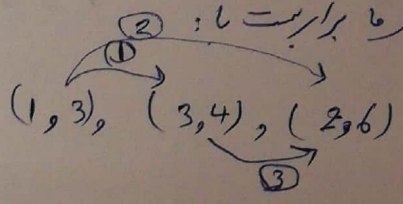
$P_{y,L} = \{(1,3), (3,4), (2,6)\}$

$P_{x,L} = \{(1,3), (2,6), (3,4)\}$

(در واقع برای هر دو بخش چپ و راست، P_L و P_{x-L} و P_{y-L} را تشکیل می‌دهیم.) و همان‌طور که می‌بینید M' جدید برابر 2 است (مختصات x راسِ وسط در آرایه P_{x-L}). اگر تقسیم کنیم به دو بخش، در هر بخش یک نقطه قرار می‌گیرد. پس بالانس است و می‌توان نقطه $(2,6)$ را به هرکدام اضافه کرد و ما به دلخواه به چپ اضافه می‌کنیم. در بخش راست آن 1 نقطه است پس کاری نمی‌کنیم. در بخش چپ آن دو نقطه است پس فاصله min در این بخش برابر $d = \sqrt{(2-1)^2+(6-3)^2} = \sqrt{10}$



حال این دو بخش را merge می‌کنیم و تنها نقطه‌هایی که مختصات x آن‌ها از $M'-d$ تا $M'+d$ است را در نظر می‌گیریم (که در اینجا 3 نقطه می‌شود). حال هر عضو را سر این بازه را برحسب y مرتب شده و چک می‌کنیم و با 7 عضو بعدی طبق مقدار y و در این بازه، فاصله آن را پیدا می‌کنیم (یکی از الگوریتم‌ها این بود که راس‌های بازه $M-d$ و $M+d$ را طبق مختصات y مرتب شده داریم و در این آرایه جابجایش کرده و هر عضو را با 7 عضو بعد خود چک می‌کنیم و min فاصله را update می‌کنیم) پس نقاط ما برابرند با:



$d_1 = \sqrt{(3-1)^2+(4-3)^2} = \sqrt{5}$

$d_2 = \sqrt{(2-1)^2+(6-3)^2} = \sqrt{10}$

$d_3 = \sqrt{(2-3)^2+(6-4)^2} = \sqrt{5}$

$\Rightarrow \min(d_1, d_2, d_3) = \sqrt{5}$

ادامه سوال ۱)

سپس برای بخش چپ هم حساب کردیم. حال کافیست که این دو بخش را merge کنیم. یادمان است که d_L و d_R را حساب کردیم (و هر دو برابر با $\sqrt{5}$ شد). باید نقاط در بازه $M-d$ و $M+d$ را در نظر بگیریم که $d=\min(d_R,d_L)$ $=\sqrt{5}$

سپس باید تمام نقاطی که مختصات x آنها در بازه $M-\sqrt{5}$ و $M+\sqrt{5}$ است را چک کنیم (که دوباره شامل همه نقاط می‌شود)

حال راس‌ها را طبق y مرتب شده داریم و مقایسه هر راس را با ۷ راس بعدی موجود در این آرایه انجام می‌دهیم و داریم:

$\{(5,1),(1,3),(4,3),(3,4),(2,6)\}$

$d_1=\sqrt{(5-1)^2+(1-3)^2}=\sqrt{20}$

$d_2=\sqrt{(5-4)^2+(1-3)^2}=\sqrt{5}$

$d_3=\sqrt{(5-3)^2+(1-4)^2}=\sqrt{13}$

$d_4=\sqrt{(5-2)^2+(1-6)^2}=\sqrt{34}$

$d_5=\sqrt{(1-4)^2+(3-3)^2}=\sqrt{9}$

$d_6=\sqrt{(3-1)^2+(4-3)^2}=\sqrt{5}$

$d_7=\sqrt{(2-1)^2+(6-3)^2}=\sqrt{10}$

$d_8=\sqrt{(3-4)^2+(4-3)^2}=\sqrt{2}$

$d_9=\sqrt{(2-4)^2+(6-3)^2}=\sqrt{13}$

$d_{10}=\sqrt{(2-3)^2+(6-4)^2}=\sqrt{5}$

(راس درانی)

و همانطور که می‌بینید، min فاصله برابر با $\sqrt{2}$ است. (دقت کنید برای هر نقطه با ۷ راس بعدی طبق y چک می‌کنیم. در اینجا چون کلا ۵ تا راس داشتیم، برای راس اول با ۴ تا، برای راس دوم با ۳ تا و راس سوم با ۲ تا و راس چهارم با ۱ تا چک کردیم. پس از اردر خطی است)

$\Longrightarrow \min d=\sqrt{2}$

2) این ایده کلی سوال به این شکل است که شبیه Merge Sort عمل می‌کنیم. یعنی در هر مرحله آرایه را به دو قسمت تقسیم می‌کنیم و برای هر کدام از بخش‌ها حل کرده (یعنی تعداد جفت عضوهایی از نوع $i<j$ و $A[i]>A[j]$ است را حساب می‌کنیم) و همچنین مثل Merge Sort، آن‌ها را نیز sort می‌کنیم. حال که باید دو بخش sort شده را با هم merge کنیم، قبل از merge کردن آن‌ها، تعداد حالت‌هایی را می‌شماریم که i در بخش راست و j در بخش چپ است $A[i]>A[j]$ (حالت‌های یک بخش را قبلا شمرده‌ایم) و بعد از شمردن می‌توانیم merge کنیم و به مراحل بالاتر برویم. در واقع سودوکد آن به این شکل است:

```
int Solve (array A, first, last) {
    if first == last
        return 0;
    int mid = (first + last) / 2
    int count_Left = Solve (A, first, mid);
    int count_right = Solve (A, mid+1, end);
**  int count_both = Count (A, first, mid, end);
    merge (A, first, mid, end);
    return (Countright + Count_left + Count_both);
}
```

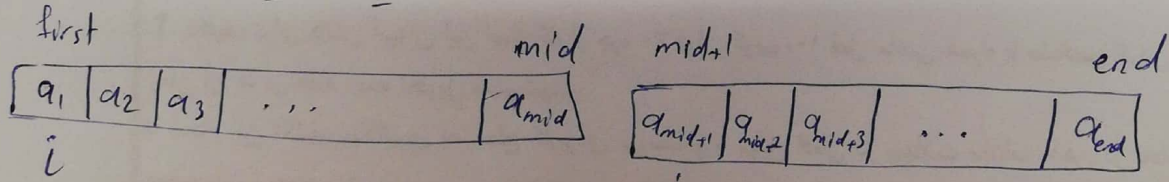
حل برای بخش چپ، پیدا کردن جواب و sort کردن آن

حل بخش راست، پیدا کردن تعداد حالت‌ها و مرتب کردن

شمردن حالت‌هایی که i در چپ و j در بخش راست است

در واقع تابع Solve برای این سوال شبیه تابع merge sort است و تنها یک تابع Count اضافی دارد.

الگوریتم تابع Count به این صورت خواهد بود: دو تا بخش sort شده داریم و متغیر counter با مقدار اولیه صفر



دو اندیس i (برای بخش چپ) و اندیس j (برای بخش راست) داریم. تا زمانی که a_j از ۳ برابر a_i کوچیکتر است، j را یک واحد جلو می‌بریم. زمانی که a_j بزرگتر مساوی $3a_i$ شد، مقدار $j-(mid+1)$ را به counter اضافه کرده و i را یک واحد جلو می‌بریم. از آنجایی که تمام عناصر a_{mid+1} تا a_{j-1} از ۳ برابر a_i کوچیکتر بودند و a_{i+1} از a_i، بزرگتر مساوی است، پس تمام عناصر قبل a_j هنوز از ۳ برابر a_{i+1} کوچیکتر هستند و لازم نیست دوباره چک شوند و حالا a_j را با a_{i+1} چک می‌کنیم. این روند را ادامه می‌دهیم (یعنی تا زمانی که a_j از ۳ برابر a_i کوچیکتر بود $\leftarrow j+=1$ و اگر بیشتر مساوی بود $\leftarrow i+=1$ و $counter = j-(mid+1)$) بعد که i به انتهای mid رسید، مقدار counter را برمی‌گردانیم) از آنجا که در هر مرحله یا اندیس i یا اندیس j اضافه می‌شوند و i حداکثر تا mid و j حداکثر تا end می‌رود پس حداکثر یک بار پیمایش داریم و پیچیدگی زمانی $O(n)$ است

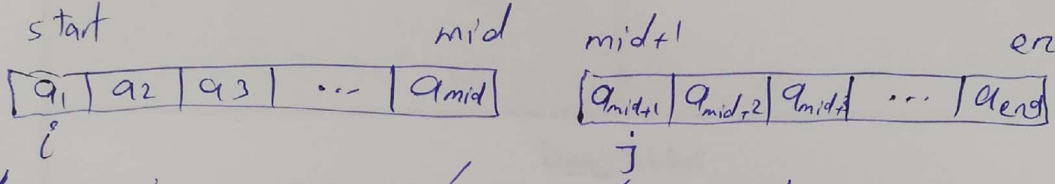
ادامه 2) پیچیدگی عملیات برابر است با:

$$T(n) = 2\left(T\left(\frac{n}{2}\right)\right) + O(n) + O(n)$$

$O(n)$ ← حل دو بخش چپ و راست — $O(n)$ ← count کردن — $O(n)$ ← merge کردن

$$\Longrightarrow T(n) = 2T\left(\frac{n}{2}\right) + O(n) \xrightarrow[\text{master}]{\text{طبق قضیه}} T(n) = O(n \log n)$$

سوال 3) برای این سوال هم مثل ایده سوال قبل عمل می‌کنیم. تابع شبیه merge sort داریم که قبل از merge کردن، عمل count را انجام می‌دهیم. پس دو بخش sort شده داریم و برای هرکدام از بخش‌ها تعداد جفت اعدادی را که $i > j$ و $a[j] < a[i]$ را حساب کردیم. حال باید برای حالتی که j در بخش راست و i در بخش چپ است، حساب کنیم. پس داریم: (متغیر counter=0 برای ذخیره تعداد)



اینبار کس ناوز را از ابتدا در هر بخش شروع می‌کنیم. تا وقتی که $a[j] < a[i]$ بود، j را یکی یکی جلو می‌کنیم و تعداد را چک می‌کنیم. اگر a_j از a_i بزرگتر ماوز شد داریم: $counter += j - (mid+1)$. به تعداد اعدادی که a_i از آنها بزرگتر بوده اضافه کردیم. حال i را یکی جلو اضافه می‌کنیم، چون a_{i+1} از a_i بزرگتر ماوز است، و تمام عناصر بین a_{mid+1} تا a_j قبل هم بزرگتر است، پس دیگر با آنها چک نمی‌کنیم و با همان a_j چک می‌کنیم و این روند را تکرار می‌کنیم تا زمانی که i به انتهای بخش چپ رسید و تمام اعداد چک شده بود. از آنجایی که در هر مرحله یا i یا j یک واحد اضافه می‌شود و محاسبات هر بخش هم از $O(1)$ است، حداکثر کل آنها را پیمایش می‌کنیم و پیچیدگی این عمل از $O(n)$ است. در انتها counter را برمی‌گردانیم. حالت پایه هم زمانی است که یک بخش قابل تقسیم شدن نباشد و یک عضو بیشتر نداشته باشد، پس صفر را return می‌کنیم در این حالت.

پیچیدگی برابر است با:

$$T(n) = 2\left(T\left(\frac{n}{2}\right)\right) + O(n) + O(n)$$

هر دو بخش sort شده ← برای count کردن ← برای merge کردن

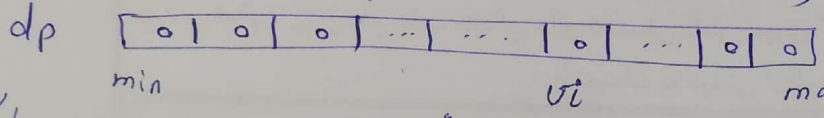
$$\Longrightarrow T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$$

۴) برای حل این سوال ابتدا ایده استفاده از dp به ذهنم رسید و بعد ایده segment tree که توسط تای مفرح شد. من هر دو روش را خواهم گفت. dp با $O(nK)$ و segment tree با $O(n \log n)$ قابل حل است. (البته با فرض اینکه قدرت اعضا نیز در اردر $O(n)$ باشند، در غیر اینصورت باید با الگوریتم پیچیده‌تر مثل map کردن، مسئله را حل کرد.)

راه اول: dp با اردر $O(nK)$

این نکته را در نظر بگیرید. هر دنباله‌رو که صعودی تا عنصر نام با مقدار v_i، درصورتی که بخواهد شامل خود v_i نیز باشد، باید عدد آخر این دنباله قبل از انتخاب v_i، در بازه $[v_i - K, v_i - 1]$ باشد، یعنی عنصر آخر زیر دنباله صعودی باید حداکثر K تا با v_i فاصله داشته باشد (کمتر باشد). حال راه حل ما این است. در هر مرحله سعی می‌کنیم طولانی‌ترین زیر دنباله تا آنجا را پیدا کنیم. یعنی در نظر داشته باشید که متغیر آرایه ذخیره‌سازی را داشته باشیم که سایز آن برابر maxVal - minVal + 1 است. یعنی:

(min قدرت حداقل می‌تواند صفر باشد)



اول همه آرایه را با مقدار اولیه صفر پر می‌کنیم

حال از ایندکس اول در صف افراد شروع می‌کنیم و مقدار v_1 را در آرایه dp به ۱ می‌گذاریم. تک تک افراد صف را یکی یکی باید چک کنیم یعنی در مرحله نام، قدرت فرد نام یعنی v_i را طبق فرمولی که می‌گم update می‌کنیم:

$$dp[v_i] = \max(dp(v_i-1), dp(v_i-2), dp(v_i-3), \dots, dp(v_i-K)) + 1$$

می‌دانیم که $dp(n)$، ماکسیمم طول دنباله‌رو صعودی است که با قدرت x پایان می‌پذیرد و می‌دانیم برای محاسبه $dp(y)$ باید بدنبال $dp(y-1)$ تا $dp(y-K)$ ها می‌باشیم که قبل از رسیدن به اندیس عدد y، حساب کردیم. از آنجایی که به همه مقدار اولیه صفر دادیم و از اول شروع کردیم و تک تک جلو رفتیم، مطمئن هستیم تمام مقادیر dp که داریم برای قبل از اندیس نام با قدرت مثلاً y است.

پس ماکزیمم $dp(y-1)$ تا $dp(y-K)$ را حساب می‌کنیم و به علاوه ۱ می‌کنیم (یعنی عنصر نام با قدرت y را هم به بزرگ‌ترین دنباله صعودی که عدد پایانی آن در بازه $y-1$ تا $y-K$ بود، اضافه کردیم.) بعد از پیمایش تمام عناصر، بزرگ‌ترین dp را گزارش می‌کنیم. (اگر در هر مرحله maxval update کنیم این کار در اردر $O(1)$ انجام می‌شود.) از آنجایی که هر عدد با قدرت v_i تنها با حداکثر K تا عنصر (یعنی $dp(v_i-1)$ تا $dp(v_i-K)$) چک می‌کنیم پس به اندازه $O(K)$ برای هر عدد و در کل $O(nK)$ برای همه اعداد خرج کردیم. حال برای پیدا کردن بزرگ‌ترین عدد در بازه $dp(v_i-1)$ تا $dp(v_i-K)$ در هر مرحله، می‌توانیم با کمک segment tree این کار را با اردر $\log K$ انجام دهیم.

ادامه ۴ صفحه بعد

ادامه سوال ۴)

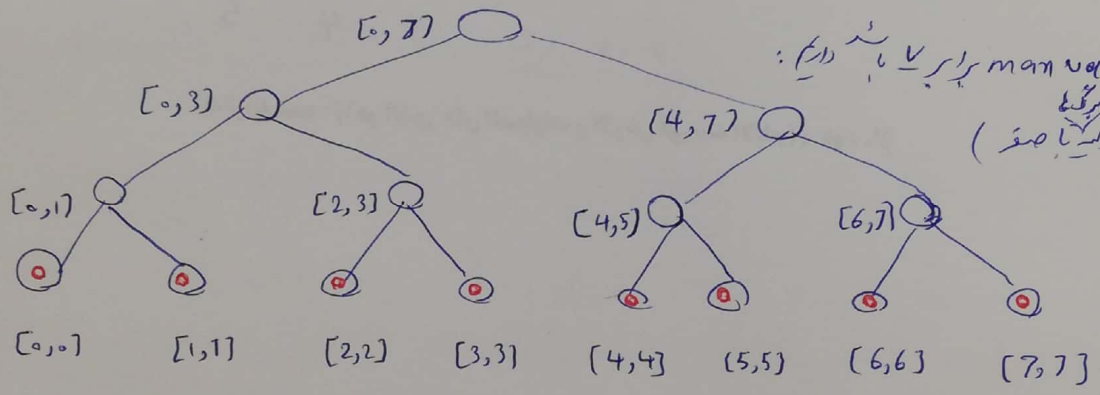
راه دوم: divide & conquer با یک segment tree: رنج

از segment tree استفاده می‌کنیم با این تفاوت که برگ‌های آن به جای اینکه اعداد باشند، قدرت افراد باشد (یعنی به جای ناحیه با ناحیه در برگ سروکار داریم). پس باید به تعداد $[min_{val}, max_{val}]$، برگ داشته باشیم (چون min حداقل می‌تواند صفر باشد، برای راحتی کار می‌توانیم بگوییم از بازه $[0, max_{val}]$) و مثل راه قبل، مقدار تمام برگ‌ها را در ابتدا صفر می‌دهیم. range query ما برابر با ماکسیم مقدار برای بازه $[i-k, i-1]$ است و این مقدار را به علاوه ۱ می‌کنیم و مقدار برگ i ام را با آن update (پیدا کردن) می‌کنیم.

باز هم مثل سوال قبل، چون از اندیس ۱ شروع کرده و تک تک جلو آمدیم، وقتی به عضر i ام رسیدیم، تمام مقادیر برای قبل عدد i ام است. (در واقع برگ i ام یک جورهایی مقدار $dp(i)$ را نگه می‌دارد). چون ارتفاع درخت segment برابر $\log n$ است، ما در range query هم در زمان $\log n$ به پاسخ می‌رسیم و برای n عدد هم $O(n \log n)$ داریم. در نهایت هم max مقدار برگ‌ها را گزارش می‌کنیم.

مثلا اگر max_{val} برابر ۷ باشد داریم:

(مقدار دهی اولیه برگ‌ها صفر)



از آنجا که range query ما هر دفعه می‌تواند در یک بست درخت باشد، range بازه‌ای که بررسی می‌کنیم نصف می‌شود و برای هر نصف ابتدا محاسبه می‌کنیم و بعد در هنگام merge کردن، تنها کافی است مقدار max آنها را بگیریم و بالاتر در درخت بفرستیم. طبق اثباتی که در ویدئو شد این کار لگاریتم ($O(\log n)$) است و از آنجا که ارتفاع درخت است در واقع (می‌دانیم که range query ما در $O(\log n)$ پاسخ می‌دهند.)

نکته این است که هر وقت از بازه $i-k$ تا $i-1$، maximum را پیدا کردیم، می‌توانیم علاوه بر ذخیره کردن مقدار maximum به علاوه ۱ در $dp[i]$، یک چیز دیگر هم ذخیره کنیم که در نهایت بدانیم که کدام عدد از نزدیک‌ترین زیر دنباله اکیدا صعودی را تشکیل می‌دهند.

۵) اولا بقلم باید به صورت $\lfloor \log_2^n \rfloor + 1$ باشد. برای اینکه دو بازه، یک عدد متمایز داشته باشیم از ایده devide&conquer استفاده می‌کنیم به این شکل که عدد وسط را ($\lceil \frac{n+1}{2} \rceil$ امین عدد مثلا) انتخاب می‌کنیم. بعد بازه سمت چپ عدد را حل می‌کنیم و بازه سمت راست عدد را نیز حل می‌کنیم (می‌دانیم که طول بازه سمت چپ، مساوی یا حداکثر یکی واحد بزرگتر از طول بازه سمت راست است) عدد‌هایی که در بازه سمت چپ استفاده شده است فرض کنیم از ۱ تا i تا است. از آنجایی که بازه سمت راست از بازه سمت چپ جداست و جداگانه است، پس می‌توانیم از همین اعداد ۱ تا i تا برای ساخت آن نیز استفاده کنیم. مثلا فرض کنید عنایت سمت چپ را در سمت راست کپی کردیم و چون طول سمت راست کمتر مساوی سمت چپ است، قطعا اگر بازه‌ای از سمت راست انتخاب کنیم به مشکل نمی‌خوریم چون مثلا سمت چپ می‌شود و ما می‌دانیم در سمت چپ هم مشکل نداشتیم. حال اگر بازه‌ای که انتخاب می‌کنیم هم از سمت راست باشد هم از سمت چپ باشد، چه کنیم؟ کافی است عدد وسط را $i+1$ بگذاریم تا مطمئن شویم که تنها یک عدد وجود دارد که در آن بازه متمایز است (اگر بازه‌ها فقط در یک سمت باشند، قبلا به صورت بازگشتی حل شده‌اند.)

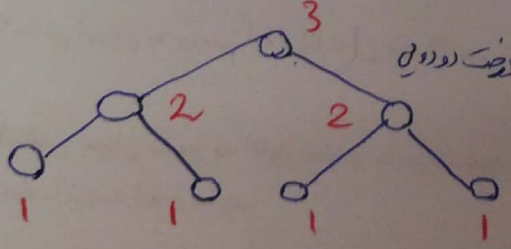
هر دفعه که به بخش تقسیم می‌کنیم، عدد وسط باید متمایز باشد و این عمل را (انتخاب عدد وسط و تقسیم به دو بخش) تا آنجایی انجام می‌دهیم که طول هر دو بخش صفر شود و فقط عدد وسط باشد. پس انگار اگر به صورت bottom_up نگاه کنیم هر دفعه حداکثر $2n+1$ برابر می‌شود و یک عدد متمایز اضافه می‌شود پس داریم:

حداکثر طول بازه	0	1	3	7	15	...	2^n-1
عدد متمایز	0	1	2	3	4	...	n

بزارید یکم دقیق‌تر بنویسم

طول بازه ممکن	$[0]$	$[1]$	$[2,3]$	$[4,7]$	$[8,15]$	...	$[2^{n-1}, 2^n-1]$
عدد متمایز	0	1	2	3	4	...	n

مثلا برای برگردان ۷ تا عدد می‌توانیم اینجوری عمل کنیم:



پیمایش in_order درخت دودویی $\rightarrow \{1,2,1,3,1,2,1\}$

پس بازه $[2^{n-1}, 2^n-1]$ را با n عدد پر کردیم، یعنی هر بازه‌ای به طول n برابر با $\lfloor \log_2^n \rfloor + 1$

$\lfloor \log \rfloor = n-1$ ✓ $\lfloor \log \rfloor = n-1$