داده ساختار هرم بیشینه (Max heap):

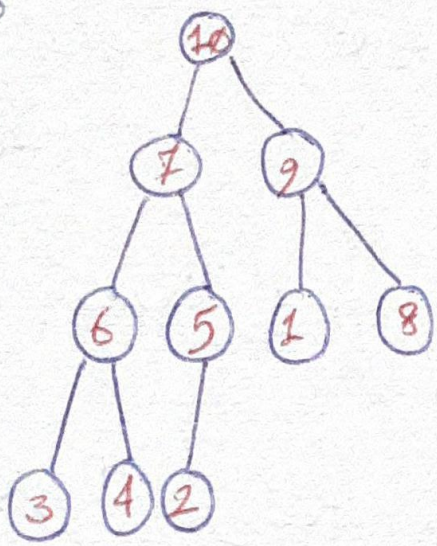
یک درخت دودویی تقریباً کامل است که ویژگی‌های زیر را دارا می‌باشد:

① هر گره، بزرگتر یا مساوی فرزندانش می‌باشد.

② برگ‌ها در سطح آخر از سمت چپ به راست پر شده است.

توالی: 1, 2, 3, 4, 5, ..., 10

مثال:



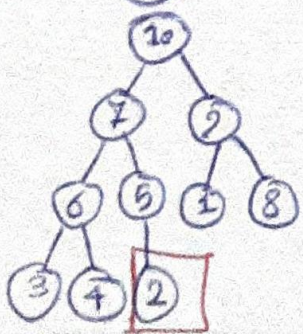
برخی نکات:

۱ در درخت Maxheap همواره عنصر بیشینه در ریشه قرار دارد.

۲ عمق درخت Maxheap برای n عنصر همواره از مرتبه $O(\log n)$ است.

۳ درخت Maxheap امکان جستجوی سریع یک کلید را فراهم نمی‌کند و جستجو می‌تواند از مرتبه $O(n)$ باشد بدین معنا که ممکن است کل نودهای درخت را پویش کنیم. (بدترین حالت)

به عنوان مثال جستجوی کلید 2 در درخت روبرو را در نظر بگیرید:



۴ در بررسی درخت BST دیدیم که اگر

① یک توالی از اعداد متمایز،

② و یک اسکلت نامتب برای آن داده شود،

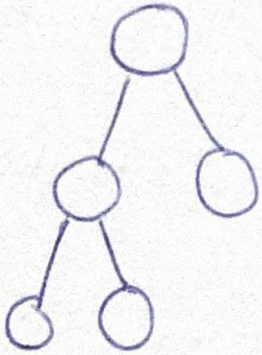
آنگاه یک مقداردهی یکتا برای نودهای درخت BST وجود دارد.

با اینحال با داشتن دو مورد بالا، نمی‌توان یک مقداردهی یکتا برای نودهای درخت Maxheap داشت.

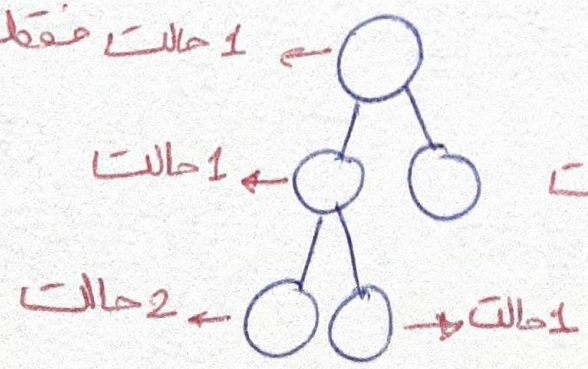
مثال: توالی روبه رو را در نظر بگیرید:

توالی: 1, 2, 3, 4, 5

همچنین اسکلت زیر را نیز برای درخت Maxheap در نظر بگیرید:



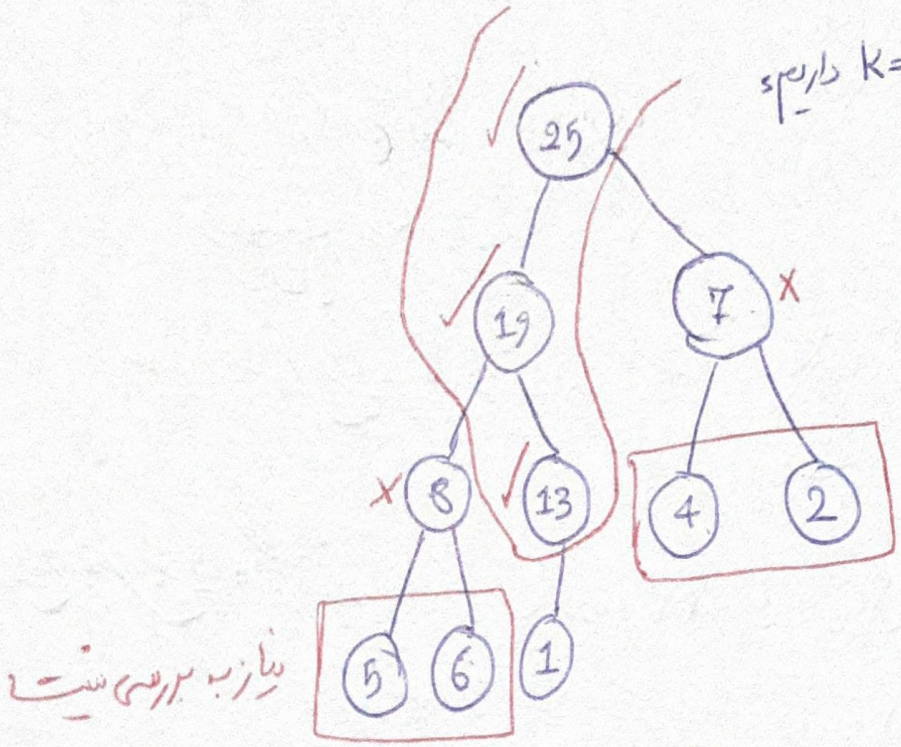
تعداد حالت هایی که می‌توانیم داشته باشیم به قرار زیر است:



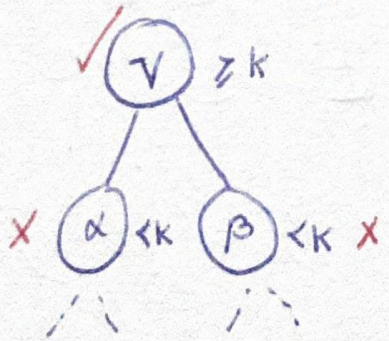
۲ تعداد کل حالات: 1 * 1 * 4 * 2 * 1 = 8

❌ درصورتیکه بخواهیم اعداد بزرگتر یا مساوی k را در یک درخت Max heap شمارش کنیم، پیچیدگی آن از مرتبه $O(2^r) = O(r)$ است جائیکه r بیانگر تعداد اعداد بزرگتر یا مساوی k در درخت Max heap است.

مثال: برای k=10 داریم:

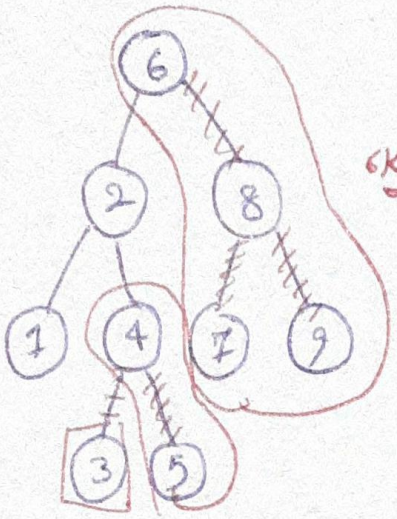


عدد 2^r در مرتبه $O(2^r)$ بیان خاطر است که در بدترین حالت همه برگ‌ها بزرگتر از k (یا مساوی) است ولی فرزندان آن کوچکتر از k هستند و باید 2^r مقایسه برای آن داشته باشیم.



اگر بخواهیم مساله‌ی جستجو را با درخت BST انجام دهیم، پیچیدگی زمان آن از مرتبه $O(\log n + T)$ است، جایگیر T بیانگر تعداد اعداد بزرگتر مساوی k است و n تعداد گره‌های درخت است.

مثال:



$\log n$: برای پیدا کردن مکان مناسب برای k، (ممکن است خود k در درخت نباشد)

T : حالت بدی و پیمایش درخت.

جمع بندی نهایی: اگر تعداد اعداد جستجو شده (T) کم باشد آنگاه درخت Max heap بهتر است و اگر این تعداد زیاد باشد پیچیدگی هر دو (درخت BST و درخت Max heap) یکسان است.

ساخت یک درخت Max heap از روی یک توالی:

راه حل ۱: ابتدا توالی داده شده را به صورت نزولی مرتب می‌کنیم، سپس نودهای درخت را سطح به سطح پر می‌کنیم.

مثال: توالی زیر را با استفاده از راه حل ۱ به یک درخت Max heap تبدیل کنید:

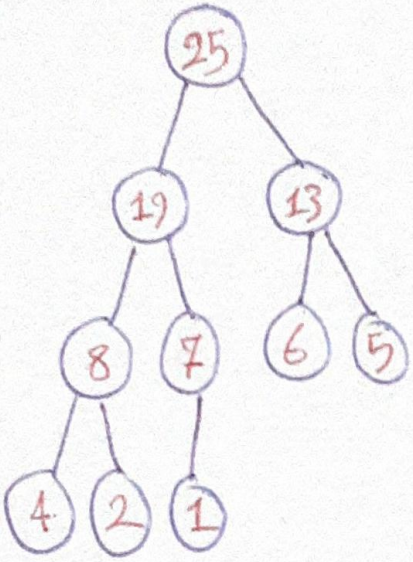
توالی: 5, 6, 1, 8, 13, 19, 4, 2, 7, 25

گام ۱: مرتب سازی نزولی ترکیبی:

~~[illegible]~~

25, 19, 13, 8, 7, 6, 5, 4, 2, 1

گام ۲: تشکیل اسکلت درخت و پر کردن سطح به سطح:



پیچیدگی زمانی راه حل ۱ از مرتبه $O(n\log n + n) = O(n\log n)$ است، چون $n\log n$ برای مرتبه سازی نزولی است و n برای پیمایش سطحی اسکلت درخت و بروزرسانی نود هاست.

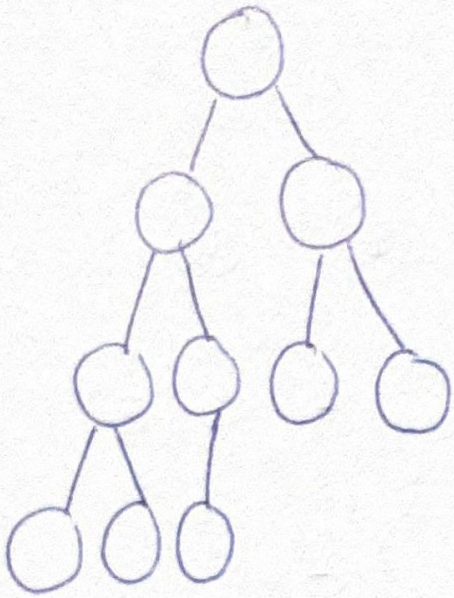
راه حل ۲: در این راه حل به دنبال آن هستیم که پیچیدگی زمانی ساخت یک درخت Max heap از روی یک آرایه را در مرتبه $O(n)$ انجام دهیم.

برای فهم ساده‌تر، راه حل ۲ را در ادامه همراه با یک مثال تشریح می‌کنیم.

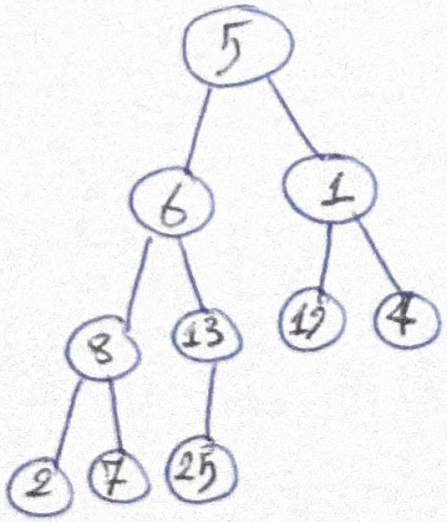
توالی زیر را به عنوان مثال در نظر بگیرید:

توالی: 5, 6, 1, 8, 13, 19, 4, 2, 7, 25

① ابتدا اسکلت درخت را می‌سازیم که شرایط ساختاری Max heap را دارا باشد.



② سپس توالی را سطح به سطح در اسکلت بالا درج می‌کنیم:

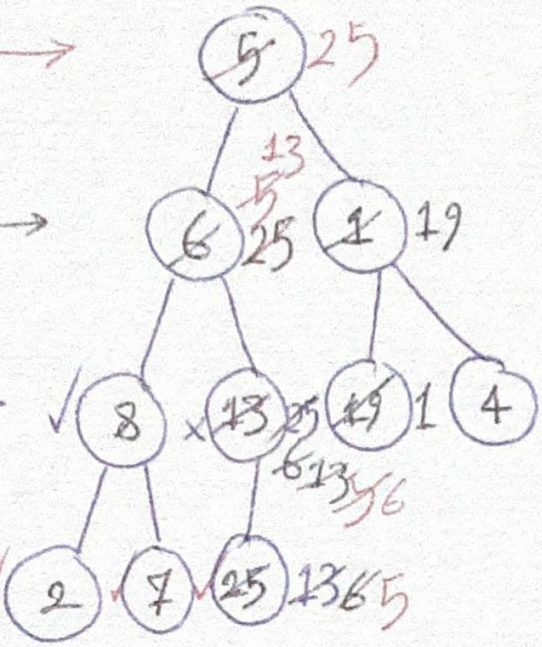


③ سپس از سطح آخر و به سمت ریشه و سطح به سطح بررسی می‌کنیم که آیا گره با فرزندان خود حقیقه Max heap بودن را دارا باشد. لازم به ذکر است که بررسی برای هر گره تا سطح آخر ادامه پیدا می‌کند.

تا سطح آخر باید چک کنیم →

تا سطح آخر باید چک کنیم →

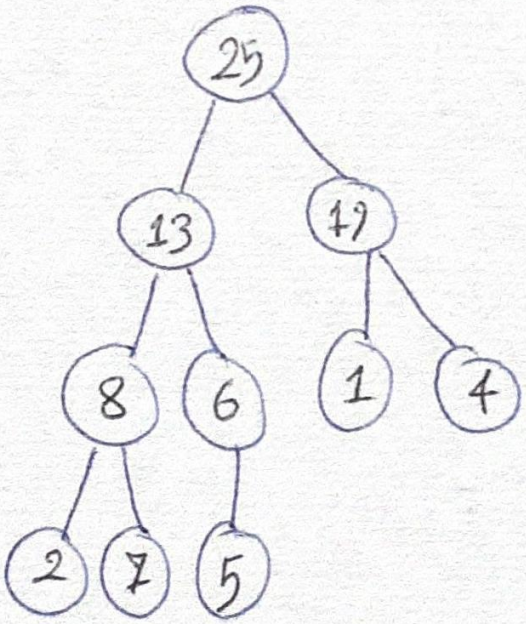
تا سطح آخر باید چک کنیم →



جهت حرکت

چون فرزند ندارند برقرار است →

درخت نهایی به صورت زیر بدست می آید:



نکته: در هر سطح، متناسب با سطحی که هستیم ممکن است تا برگ مقایسه و جابجایی شکل گیرد.