In [1]:
```
# setup
from IPython.core.display import display,HTML
display(HTML('<style>.prompt{width: 0px; min-width: 0px; visibility: collapse}</style>'))
display(HTML(open('rise.css').read()))

# imports
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(style="whitegrid", font_scale=1.5, rc={'figure.figsize':(12, 6)})
import time
```

CMPS 2200

Introduction to Algorithms

Recurrences - The Brick Method

Recurrences can be more general, so how do we get a handle on asymptotic runtimes when the recursion is really complicated?

Key Idea: The branching properties of the recursion tree determine work at each level and the number of leaves.

The **brick method** gives a way to derive asymptotic runtimes by looking at the relationships between parent and child nodes in the recursion tree. This way we only need to worry about the costs of the root and the leaves.

> In the tree method, if the costs grow or decay geometrically across levels, then we need only consider the cost of the root (**decay**), or the total cost of the leaves (**growth**).
>
> If there is no geometric growth or decay then we can calculate the cost of the worst level (often either the root or leaves) and multiply it by the number of levels.

This leads to three cases:

1. **root dominated**
2. **leaf dominated**
3. **balanced**

> Conveniently, to distinguish these three cases we need only consider the cost of each node in the tree and how it relates to the cost of its children.

The value of n decreases geometrically as we collect the terms in our recurrences. We'll make use of bounds for geometric series. For any $\alpha > 1$:

$$\sum_{i=0}^{n} \alpha^i = \frac{\alpha}{\alpha - 1} \cdot \alpha^n$$

For $\alpha < 1$:

$$\sum_{i=0}^{\infty} \alpha^i < \frac{1}{1-\alpha}$$

Root-dominated

For a node v in the recursion tree, let $C(v)$ denote its cost and $D(v)$ denote its children.

A recurrence is **root-dominated** if for all v, there is an $\alpha > 1$ such that:

$$C(v) \geq \alpha \sum_{u \in D(v)} C(u)$$

The cost of a root dominated recurrence is $O(C(r))$ if r is the root.

This is because the cost reduces geometrically as we go toward the leaves, and the total cost bounded by $\alpha/(\alpha - 1)$ times $C(r)$.

$$W(n) = 2W\left(\frac{n}{2}\right) + n^2$$

$C(\text{root}) = n^2$

$C(\text{level } 1) = \left(\frac{n}{2}\right)^2 + \left(\frac{n}{2}\right)^2 = \frac{n^2}{2}$

Cost has **decreased** by a factor of two when descending one level in the tree.

so, if $\alpha \leftarrow 2$:

$C(v) \geq \alpha \sum_{u \in D(v)} C(u)$

$n^2 \geq 2 * \frac{n^2}{2}$

$n^2 \geq n^2$

Because the cost is asymptotically dominated by the **root**, we need to only consider its cost: $O(n^2)$.

Leaf-dominated

A recurrence is **leaf-dominated** if for all v, there is an $\alpha > 1$ such that:

$$C(v) \leq \frac{1}{\alpha} \sum_{u \in D(v)} C(u)$$

If we have L leaves in the recursion tree, the cost of a leaf dominated recurrence is $O(L)$.

This is because the cost increases geometrically as we go toward the leaves, and the total cost is bounded by $\alpha/(\alpha - 1)$ times $c_b \cdot L$.

$$W(n) = 2W\left(\frac{n}{2}\right) + \sqrt{n}$$

$C(\text{root}) = \sqrt{n}$

$C(\text{level 1}) = \sqrt{\left(\frac{n}{2}\right)} + \sqrt{\left(\frac{n}{2}\right)} = 2\frac{\sqrt{n}}{\sqrt{2}} = \sqrt{2}\sqrt{n}$

Cost has **increased** by a factor of $\sqrt{2}$ when descending one level in the tree.

so, if $\alpha \leftarrow \frac{1}{\sqrt{2}}$:

$C(v) \leq \frac{1}{\alpha} \sum_{u \in D(v)} C(u)$

$\sqrt{n} \leq \frac{1}{\sqrt{2}} * \sqrt{2}\sqrt{n}$

$\sqrt{n} \leq \sqrt{n}$

Because the cost is asymptotically dominated by the **leaves** (i.e., lowest level of the tree), we need to only consider their cost:

- Cost of each leaf is 1 (since $\sqrt{1} == 1$).
- Depth of tree is $\lg n$ (since we divide by 2 at each level)
- Number of leaves of a binary tree with depth $\lg n$ is $2^{\lg n} = n$

So, final cost is $n * 1 = O(n)$

Balanced

A recurrence is **balanced** when every level of the recursion tree has the same asymptotic cost. In this case, the recurrence is *number of levels* times *maximum cost per level*.

$O(D(r) \log n) = O(L \log n)$.

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

$C(\text{root}) = n$

$C(\text{level 1}) = \left(\frac{n}{2}\right) + \left(\frac{n}{2}\right) = n$

Cost has **remained the same** when descending one level in the tree.

So, total cost is *number of levels* times *maximum cost per level*.

- Number of levels = $\lg n$
- Maximum cost per level = n (last level: n leaves with cost 1 each; first level: one node with cost n)
- $O(n \lg n)$

Let's look at some examples:

$$W(n) = 2W(n/2) + \sqrt{n}$$

$$W(n) = 3W(n/2) + n$$

$$W(n) = 2W(n/3) + n$$

$$W(n) = 3W(n/3) + n$$

Do you see a way to count the number of leaves in the recursion tree?

> Useful observation: If we have a recurrence of the form $W(n) = aW(n/b) + f(n)$, then the number of leaves is $O(a^{\log_b n})$, or equivalently, $O(n^{log_b a})$.

$$W(n) = 2W(n/2) + \sqrt{n}$$

This is leaf-dominated, so it is $O(n)$.

$$W(n) = 3W(n/2) + n$$

This is leaf-dominated, so it is $O(n^{\log_2 3})$.

$$W(n) = 2W(n/3) + n$$

This is root-dominated, so it is $O(n)$.

$$W(n) = 3W(n/3) + n$$

This is balanced, so it is $O(n \log n)$.

More examples (some trickier than others):

$$W(n) = W(n-1) + n$$

$$W(n) = \sqrt{n}W(\sqrt{n}) + n^2$$

$$W(n) = W(\sqrt{n}) + W(n/2) + n$$

$$W(n) = W(n/2) + W(n/3) + 1$$

$$W(n) = W(n-1) + n$$

This is actually a balanced recurrence since every level has the same cost and there are n levels - the recurrence is $O(n^2)$.

$$W(n) = \sqrt{n}W(\sqrt{n}) + n^2$$

This is root-dominated so it is $O(n^2)$.

$$W(n) = W(\sqrt{n}) + W(n/2) + n$$

This is root-dominateed so it is $O(n)$.

$$W(n) = W(n/2) + W(n/3) + 1$$

This recurrence is a little tricky - while it is leaf-dominated we need to calculate the number of leaves. The book has a derivation of this.