Exercises

1. (a) Let $a_i = T(n) = T(2^i)$, $n = 2^i$. Base case: $n = 1 \Rightarrow 1 = 2^i, i = 0$.

$$\begin{aligned}
T(n) &= a_i \\
&= 2a_{i-1} + 2^i \\
&= 2(2a_{i-2} + 2^{i-1}) + 2^i \\
&= 4a_{i-2} + 2 \cdot 2^i \\
&= 4(2a_{i-3} + 2^{i-2}) + 2 \cdot 2^i \\
&= 8a_{i-3} + 3 \cdot 2^i \\
&= 2^j a_{i-j} + j \cdot 2^i \\
&...j = i \\
&= 2^i a_0 + i \cdot 2^i \\
&= (2 + i)2^i \\
&= (2 + log(n))n
\end{aligned}$$

   (b)

$$\begin{aligned}
a_i &= 2a_{i-1} + 2 \cdot 2^i \\
&= 2^i a_0 + 2i2^i \\
&= (1 + 2i)2^i \\
&= (1 + 2log(n))n
\end{aligned}$$

2. b needs to be a constant for the Master Theorem to be applicable.

$$\begin{aligned}
T(n) &= T(n-1) + n \\
&= T(n-2) + T(n-1) + n - 1 + n \\
&... \\
&= T(n - (n-1)) + (n - (n-2)) + ... + n \\
&= 1 + 2 + ... + n \\
&= n(1 + n)/2
\end{aligned}$$

3. (a) Use the Master Theorem. $a = 1$, $b = 3$, $d = 2$, $a < b^d \Rightarrow O(n^2)$.
   (b) $a = 2$, $b = 2$, $d = 1$, $a = b^d \Rightarrow O(nlog(n))$.
   (c) Guess $T(n) = O(n)$. To prove: $\exists c, n_0 \ni T(n) \leq cn$, $\forall n \geq n_0$.
   Base cases:$1 \leq cn, \forall n \leq 4$.
   Induction hypothesis: $\forall\ k = 0, 1, ..., n-1$, we have $T(k) \leq ck$.

For $k = n$,

$$\begin{aligned}
T(n) &= T(n/2) + T(n/4) + n \\
&\leq cn/2 + cn/4 + n \\
&= (3c/4 + 1)n \\
&\text{which we require} \\
&\leq cn
\end{aligned}$$

Therefore we pick $c \geq 4$, which makes the base cases and the induction correct. So $T(n) = O(n)$.

Problems

---

1. Let $n = 2^i$, $T(n) = T(2^i) = a_i$.

$$\begin{aligned}
a_0 &= a_1 = 1 \\
a_i &= 2a_{i-1} + \frac{2^i}{i} \\
&= 4a_{i-2} + 2^i(\frac{1}{i-1} + \frac{1}{i}) \\
&= 4(2a_{i-3} + \frac{2^{i-2}}{i-2}) + 2^i(\frac{1}{i-1} + \frac{1}{i}) \\
&= 8a_{i-3} + 2^i(\frac{1}{i-2} + \frac{1}{i-1} + \frac{1}{i}) \\
&... \\
&= 2^j a_{i-j} + 2^i \sum_{k=i}^{i-j+1} \frac{1}{k} \\
&...j = i - 1 \\
&= 2^{i-1} a_1 + 2^i \sum_{k=i}^{2} \frac{1}{k} \\
&= 2^i(\frac{1}{2} - 1 + \sum_{k=i}^{1} \frac{1}{k}) \\
&= n(-\frac{1}{2} + \Theta log(i)) \\
&= \Theta(nlog(log(n)))
\end{aligned}$$

2. Base case: $T(1) \geq c \cdot 1 \cdot log(1) = 0$, $\forall c \in \Re$.
   Induction hypothesis: $T(k) \geq c \cdot n \cdot log(n)$, for $k = 1, 2, ..., n-1$.

For $k = n$, we have

$$\begin{aligned}
T(n) &= 2T(\lceil n/2 \rceil) + n/2 \\
&\geq 2c\lceil n/2 \rceil log\lceil n/2 \rceil + n/2 \\
&\geq cn(logn - log2) + n/2 \\
&= cnlogn - cn + n/2 \\
&= cnlogn + (1/2 - c)n \\
&\text{which we require} \\
&\geq cnlogn
\end{aligned}$$

The induction is correct if $0 < c \leq 1/2$. So $T(n) = \Omega(nlogn)$.

3. (a)
```
def naive(bids):
  def getDaySpan(bids):
    mina = None
    maxb = None
    for i in bids:
      if mina is None or i[0] < mina:
        mina = i[0]
      if maxb is None or i[1] > maxb:
        maxb = i[1]
    return mina, maxb

  minDay, maxDay = getDaySpan(bids)
  dayMaxFish = []
  for day in range(minDay, maxDay):
    maxFish = 0
    for bid in bids:
      if day >= bid[0] and day < bid[1] and bid[2] > maxFish:
        maxFish = bid[2]
    dayMaxFish.append((day, maxFish))
  dayMaxFish.append((maxDay, 0))
  return dayMaxFish
```

(b) What doesn't work: Sort by fish amount and greedily allocate high-pay requests first. Even though greedy assignment conceptually takes O(1) for each bid, checking if days are open takes O(n). So overall it's still $O(n^2)$.

What works: divide and conquer plus linear-time merge. Complexity follows mergesort: $T(n) = 2T(n/2) + O(n)$. Correct by construction and can be proven by induction.

```
def merge(pi, pj):
  out = [(-float('Inf'), 0)]
  i, j = 0, 0
  while (i < len(pi) and j < len(pj)):
    if pi[i][0] < pj[j][0]:
      if (pi[i][1] > out[-1][1]):
        out.append(pi[i])
      i += 1
    else:
      if (pj[j][1] > out[-1][1]):
        out.append(pj[j])
      j += 1
```

```
  if i < len(pi):
    out += pi[i:]
  if j < len(pi):
    out += pj[j:]
  return out

def better(bids):
  if len(bids) == 1:
    bid = bids[0]
    return [(bid[0], bid[2]), (bid[1], 0)]

  bidi = bids[:len(bids)//2]
  bidj = bids[len(bids)//2:]
  ploti = better(bidi)
  plotj = better(bidj)
  return merge(ploti, plotj)
```