Problem 10.2.3

Task 1

Recurrence for *square_root* :

(a) The size of the input is the value of num because *square_root* is just a wrapper for srhelp, whose recurrence's input is $ub - lb$. Since in square-root srhelp is applied to $0(= lb)$ and $num(= ub)$, the relevent input is just $num - 0 = num$.
(b) When $num = 0$, the amount of work done in srhelp is a constant k_0, when $num = 1$, the amount of work done is a constant k_1, and when $2 \leq num \leq 5$, the amount of work done is a constant k_2. In these base cases, srhelp does not need to recur, though there are three base cases in srhelp, so they all do a different amount of constant work. Since I am just looking for an upper bound, I will collapse these cases as they differ only by a constant and call the amount of work done at this step k.
(c) Let the amount of work done in srhelp in the recursive case be l_1 if the average of the bounds is less than the square root and l_2 if it is greater. Since I am just looking for an upper bound, I will collapse these cases as they differ only by a constant and call the amount of work done at this step l.
(d) There is one recursive call made at each step, on $(n/2)$. Since this is just an upper bound it does not make a difference that $(n/2)$ is rounded down when n is odd.

The recurrence relation for *square_root*, $S(n)$ is as follows:

$$S(n) \leq \begin{cases} k & \text{if } n \leq 5 \\ l + S(n/2) & \text{otherwise} \end{cases}$$

Recurrence for *super_power* :

(a) The size of the input is the value of y.
(b) Let k equal the amount of work done in the base case where $y = 0$.
(c) Let l_1 equal the amount of work done in the recursive step when y is even and l_2 equal the amount of work in the recurseive step when y is odd. Since I am only looking for an upper bound, I will collapse these cases as they only differ by a constant and call the amound of work done at this step l.
(d) There is one recursive call made at each step, on $(y/2)$. Since this is just an upper bound it does not make a difference that $(y/2)$ is rounded down when y is odd.

Thus the recurrence for *super_power* should be:

$$P(n) \leq \begin{cases} k & \text{if } n = 0 \\ l + P(n/2) & \text{otherwise} \end{cases}$$

This recurrence is exactly the same as $S(n)$ except for the values of the input that are caught by the base case. Since this is an upper bound, I will use the relation P(n) to bound the runtime of both procedures.

Task 2

The table below is used to calculate the closed form of the recurrence $P(n)$ (assuming that n is a power of 2, and that $1/2 = 0$, which is the case using ocaml int division):

level	I/P Size	cost per node	# of nodes	level cost
1	n	l	1	l
2	$n/2$	l	1	l
3	$n/4$	l	1	l
...	...	...	...	...
$(\log_2 n) + 1$	1	l	1	l
$(\log_2 n) + 2$	0	k	1	k

So the total cost is the sum of all the level costs.

$$\begin{aligned} P(n) &= k + \sum_{i=1}^{(\log_2 n)+1} l \\ &= k + l((\log_2 n) + 1) \\ &= k + l + l(\log_2 n) \end{aligned}$$

Problem 10.3

Informally, the runtime of *kth* is on average linear in n, where n is the length of the list. At every recursive call it performs some constant work and some linear work in n. The linear work is at best a single call of *filter* and *length* (on the filtered list which is on average length $n/2$) and at worst two calls of *filter* and a call of *length*. Nonetheless, all this work is at most $3n$. The proceeding recursive calls are lists of approximately half the length of the list at the preceding call. Therefore, the runtime is about $3n + \frac{3}{2}n + \frac{3}{4}n + ...$, an series which just sums to 6n. This runtime is an element $O(n)$.