(a) i) Die Wahrscheinlichkeit dass getMaybePrime() eine Primzahl ausgibt ist ϵ. testPrime(N) gibt immer Primzahl wenn N eine Primzahl ist.
$\Rightarrow$ Die Wahrscheinlichkeit, dass eine Primzahl ausgegeben wird in Schritt 3 ist ϵ

ii) Die Wahrscheinlichkeit dass getMaybePrime() eine zahl ausgibt welche kein Primzahl ist: $1-\epsilon$. testPrime(N) gibt mit wahrscheinlichkeit $\frac{1}{2}$ Primzahl aus, wenn N keine primzahl ist. Insgesamt rufen wir testPrime(N) k mal auf.
$\Rightarrow$ Die Wahrscheinlichkeit, dass eine Zahl, die keine Primzahl ist, ausgegeben wird in Schritt 3 ist: $\left(\frac{1}{2}\right)^k(1-\epsilon)$

iii) Es wird nichts ausgegeben wenn getMaybePrime() eine zahl ausgibt welche kein prime ist und testPrime(N) mindestens einmal "keine Primzahl" ausgibt.
$\Rightarrow (1-\epsilon)(1-\left(\frac{1}{2}\right)^k)$

(b) Die Wahrscheinlichkeit dass wir beim i-ten iteration eine Zahl ausgeben welche keine Primzahl ist:
i=1: $\left(\frac{1}{2}\right)^k(1-\epsilon)$
i=2: $(1-\epsilon)(1-\left(\frac{1}{2}\right)^k)\cdot\left(\frac{1}{2}\right)^k(1-\epsilon)$
i=3: $(1-\epsilon)(1-\left(\frac{1}{2}\right)^k)^2\cdot\left(\frac{1}{2}\right)^k(1-\epsilon)$
...
i=m: $(1-\epsilon)(1-\left(\frac{1}{2}\right)^k)^m\cdot\left(\frac{1}{2}\right)^k(1-\epsilon)$
$\Rightarrow$ Pr[Fail] = Pr[Not returning in the previous iteration] $\cdot$ Pr[returning in an iteration]
$\Rightarrow$ Die Wahrscheinlichkeit dass der Algorithmus eine Zahl ausgibt die keine Primzahl ist, ist die Summe der Wahrscheinlichkeiten eine Zahl auszugeben die keine Primzahl ist über alle iterationen i.e:

$$P_{Fail}=\left(\tfrac{1}{2}\right)^k(1-\epsilon)\sum_{i=0}^{\infty}\big((1-\epsilon)(1-\left(\frac{1}{2}\right)^k\big)^i$$

Mithilfe der Geometrische Reihe können wir den Formel in geschlossener form bringen:

$$P_{Fail}=\left(\tfrac{1}{2}\right)^k(1-\epsilon)\sum_{i=0}^{\infty}\big((1-\epsilon)(1-\left(\frac{1}{2}\right)^k\big)^i=\frac{\left(\frac{1}{2}\right)^k(1-\epsilon)}{1-\big((1-\epsilon)(1-\left(\frac{1}{2}\right)^k\big)}$$

(c) Wir setzen $\delta=P_{Fail}\rightarrow 1-\delta=P_{Success}=\epsilon$
$\Rightarrow$ Wir müssen zeigen dass für den gegebenen k, $1-\delta\geq\epsilon$ folgt. Wir vereinfachen zuerst den Zähler von P_{Fail}:
$(\frac{1}{2})^k(1-\epsilon)=(\frac{1}{2})^{log_2(\epsilon^{-1}-1)+log_2(\delta^{-1}-1)}\cdot(1-\epsilon)=(\frac{1}{2})^{log_2(\epsilon^{-1}-1)}\cdot(\frac{1}{2})^{log_2(\delta^{-1}-1)}\cdot(1-\epsilon)$
$=\frac{1}{\epsilon^{-1}-1}\cdot\frac{1}{\delta^{-1}-1}\cdot(1-\epsilon)=\frac{1}{\frac{1}{\epsilon}-1}\cdot\frac{1}{\frac{1}{\delta}-1}\cdot(1-\epsilon)=\frac{1}{\frac{1-\epsilon}{\epsilon}}\cdot\frac{1}{\frac{1-\delta}{\delta}}\cdot(1-\epsilon)=\frac{\epsilon}{1-\epsilon}\cdot\frac{\delta}{1-\delta}\cdot(1-\epsilon)=\frac{\epsilon\delta}{1-\delta}$
Wir vereinfachen nun den Nenner:
$1-\big((1-\epsilon)(1-\left(\frac{1}{2}\right)^k\big)=1-\big((1-\epsilon)-\left(\frac{1}{2}\right)^k(1-\epsilon)\big)=1-\big((1-\epsilon)-\frac{\epsilon\delta}{1-\delta}\big)=1-\frac{(1-\epsilon)(1-\delta)-\epsilon\delta}{1-\delta}=1-\frac{1-\epsilon-\delta}{1-\delta}\geq$ $1-\frac{1-\epsilon-\delta}{(1-\epsilon)(1-\delta)}=\frac{(1-\epsilon)(1-\delta)-1+\epsilon+\delta}{(1-\epsilon)(1-\delta)}=\frac{\epsilon\delta}{(1-\epsilon)(1-\delta)}$
$\Rightarrow P_{Fail}\leq\frac{\frac{\epsilon\delta}{1-\delta}}{\frac{\epsilon\delta}{(1-\epsilon)(1-\delta)}}=1-\epsilon$
$\Rightarrow \epsilon\leq 1-P_{Fail}=P_{Success}=1-\delta$

(d) Für eine Iteration haben wir eine Laufzeit von $\mathcal{O}(kB^3+B)=\mathcal{O}(kB^3)$. In Teilaufgabe c) ist ein k gegeben welche mit ein erfolgswahrscheinlichkeit $1-\delta$ eine Primzahl ausgibt.
$\Rightarrow \mathcal{O}(kB^3)=\mathcal{O}((log_2(\epsilon^{-1}-1)+log_2(\delta^{-1}-1))B^3)=\mathcal{O}((log\epsilon^{-1}+log\delta^{-1})B^3)=\mathcal{O}(B^3(logB+log\delta^{-1}))$
Die Anzahl Iterationen ist ein geometrische verteilung wobei $p=1-(1-\epsilon)(1-\left(\frac{1}{2}\right)^k)$ ist die wahrscheinlichkeit dass unser algorithmus terminiert d.h unser Erwartungswert ist $\frac{1}{p}=\frac{1}{(\frac{1}{2})^k+\epsilon-\epsilon(\frac{1}{2})^k}\in\mathcal{O}(\frac{1}{\epsilon})=\mathcal{O}(B)$
$\Rightarrow$ Unser Algorithmus hat eine Laufzeit von $\mathcal{O}(B\cdot B^3(logB+log\delta^{-1}))=\mathcal{O}(B^4(logB+log\delta^{-1}))$